Ein integrierter Softwareansatz zur interaktiven Exploration und Steuerung von Strömungssimulationen auf Many-Core-Architekturen

Von der
Fakultät Architektur, Bauingenieurwesen und Umweltwissenschaften
der Technischen Universität Carolo-Wilhelmina
zu Braunschweig

zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)
genehmigte

Dissertation

von
Jan Linxweiler
aus Buchholz in der Nordheide

Eingereicht am	21. Juni 2011
Mündliche Prüfung am	15. November 2011
Berichterstatter	Prof. Dr.-Ing. habil. Manfred Krafczyk
	Prof. Dr.-Ing. habil. Marcus Magnor

2011

The function of good software is to make the complex appear to be simple.

- Grady Booch

Vorwort und Danksagung

Diese Arbeit enstand während meiner mehrjährigen Tätigkeit am Institut für rechnergestützte Modellierung im Bauingenieurwesen der Technischen Universität Braunschweig. Sie wurde in Teilen durch ein Promotionsstipendium der IT Research Division der Firma NEC Europe Ltd. in St. Augustin gefördert.

Mein besonderer Dank gilt meinem Doktorvater Herrn Prof. Dr. habil. Manfred Krafczyk für das entgegengebrachte Vertrauen und das Ermöglichen dieser Promotion sowie seine großzügige fachliche Unterstützung.

Bei Herrn Prof. Dr. habil. Magnor möchte ich mich für die Übernahme des Zweitgutachtens bedanken. Herrn Prof. Dr. Peil und Frau Prof. Dr. Langer danke ich für die freundliche Bereitschaft, die Aufgaben des Prüfers und der Vorsitzenden der Prüfungskommission zu übernehmen.

Bedanken möchte ich mich auch bei meinem Betreuer seitens der Firma NEC Herrn Dr. Jörg Bernsdorf für die angenehme und engagierte Zusammenarbeit. Die Aufenthalte in St. Augustin werde ich in freudiger Erinnerung behalten. Des Weiteren danke ich der IT Research Division der Firma NEC für die finanzielle Unterstützung meiner Forschungstätigkeit.

Ein großer Dank gilt den Mitarbeiterinnen und Mitarbeitern am iRMB, deren Unterstützung maßgeblich zu dem Gelingen der Arbeit beigetragen hat. Insbesondere bedanke ich mich bei meinem Kollegen Dr. habil. Jonas Tölke für die enge Zusammenarbeit auf dem Gebiet der GPU-basierten Strömungssimulation. Weiterhin danke ich Dr. Sören Freudiger und Dr. Sebastian Geller, die mich schon zu meiner studentischen Zeit begleitet und mir zahlreiche Facetten der Bauinformatik näher gebracht haben; bei Maik Stiebler, Dr. Christian Janßen und Dr. Martin Geier, die mir stets eine große Hilfe bei allen Fragen numerischer Natur waren und bei Dr. Sebastian Bindick und Jan Hegewald für die fruchtbaren Diskussionen über aktuelle Themen der Softwareentwicklung.

Des Weiteren danke ich Konstantin Kucher und Sascha Lity, die mich als wissenschaftliche Hilfskräfte bei der Implementierungsarbeit unterstützt haben.

Einen großen Dank möchte ich auch an alle Korrekturleser richten, die sehr gewissenhaft dazu beigetragen haben, das Manuskript von Fehlern zu bereinigen.

Mein größter Dank gilt jedoch meiner Familie und meinen Freunden. Insbesondere möchte ich mich bei meinen Eltern dafür bedanken, dass sie mir mein Studium ermöglichten, mich durch alle Höhen und Tiefen begleiteten und mir dabei stets Rückhalt gaben. Gleichfalls danke ich meiner Freundin Saskia, die für meine Forschungstätigkeit in besonderem Maße Verständnis und Geduld aufbrachte.

Jan Linxweiler, im Juni 2011
linxweiler@irmb.tu-bs.de

Zusammenfassung

Traditionell werden numerische Strömungssimulationen (CFD) (*Computational Fluid Dynamics*) in einer zyklischen Sequenz autonomer Teilschritte durchgeführt. In der Regel wird das Verfahren dabei mit der sogenannten *Pre-Processing*-Phase begonnen, in der die Geometrie des Strömungsgebiets ebenso wie Anfangs- und Randbedingungen definiert werden. Anschließend wird die Simulation so lange durchgeführt, bis die Lösung konvergiert (bei stationären Strömungen) oder aber eine festgelegte Zeitdauer erreicht ist (für transiente Phänomene). Da CFD-Simulationen oft mit einem hohen Rechenaufwand verbunden sind, werden sie zumeist auf Hochleistungsrechnern durchgeführt. Die Verwaltung der Ressourcen wird auf diesen Systemen in der Regel von speziellen Management-Systemen (*Queueing*-System) übernommen. Diese nehmen die Berechnungsaufträge entgegen und führen sie aus, sobald die erforderlichen Ressourcen verfügbar werden. Die Laufzeit der Simulation kann dann je nach Systemgröße Stunden, Tage oder sogar länger betragen, bis die Ergebnisdaten schließlich im Dateisystem vorliegen. Für die anschließende Analyse und Auswertung müssen diese Ergebnisse in der Regel zeitaufwändig auf den Arbeitsplatzrechner des Anwenders transferiert werden. In diesem sogenannten *Post-Processing*-Schritt können Fehler und Modifikationen identifiziert werden, die in der nächsten Iteration Berücksichtigung finden. Besonders hervorzuheben ist, dass für jede Änderung an den Parametern der gesamte Prozess zu wiederholen ist. Eine Benutzerinteraktion ist somit nur beschränkt möglich und findet ausschließlich im *Pre-Processing* bzw. in der Modifikationsphase statt. Es verwundert somit nicht, dass seitens der Wissenschaftler schon lange der Wunsch nach mehr Interaktion mit laufenden Simulationen besteht. Seit dem maßgeblichen Report der *National Science Foundation* (NSF) im Jahre 1987 wurden daher neue Formen der wissenschaftlichen Visualisierung entwickelt, die sich grundlegend von den traditionellen Verfahren unterscheiden. Insbesondere hat der sogenannte *Computational Steering*-Ansatz reges Interesse der wissenschaftlichen CFD-Community bewirkt. Damals wie heute ist die Anwendung des Verfahrens jedoch eher die Ausnahme denn die Regel. Ursächlich dafür sind zu großen Teilen die Komplexität und Restriktionen traditioneller Hochleistungssysteme. Im Rahmen dieser Arbeit wird daher als Alternative zu dem traditionellen Vorgehen die immense Leistungsfähigkeit moderner Grafikkartengenerationen für die Berechnungen herangezogen. Das sogenannte GPGPU-Computing (*General Purpose Graphics Processing Unit*) eignet sich insbesondere für die Anwendung der Lattice-Boltzmann-Methode (LBM) im Bereich numerischer Strömungssimulationen. Auf Grundlage des LBM-Verfahrens wird im Rahmen dieser Arbeit prototypisch eine interaktive Simulationsumgebung basierend auf dem *Computational Steering*-Paradigma entwickelt, das alle Prozesse zur Lösung von Strömungsproblemen innerhalb einer einzelnen Anwendung integriert. Der Einsatz der Grafikprozessoren (GPUs) bietet dabei zahlreiche Vorteile gegenüber konventionell verteilten Ansätzen. Durch die Konvergenz der hohen massiv parallelen Rechenleistung der GPUs und der Interaktionsfähigkeiten in einer einzelnen Anwendung kann beispielsweise eine erhebliche Steigerung der Qualität der Anwendung insbesondere in Hinblick auf die Benutzerfreundlichkeit erzielt werden. Dabei ist es durch Einsatz mehrerer GPUs möglich, eine Rechenleistung zu erzielen, die es für geeignete Turbulenzmodelle mit hinreichender Geschwindigkeit erlaubt, eine Näherungslösung für dreidimensionale Strömungsprobleme mit pra-

xisrelevanter Problemgröße zu berechnen und gleichzeitig eine interaktive Manipulation und Exploration des Strömungsgebiets zur Laufzeit zu ermöglichen. Da dieser Ansatz nicht notwendigerweise den Einsatz der ansonsten üblichen Netzwerkkommunikation erfordert, profitiert insbesondere die Responsivität der Anwendung in erheblichem Maße davon, dass die netzwerktypische Bandbreitenlimitierung und Latenz vermieden werden. Zusätzlich kann auf diese Weise die Komplexität der Anwendung deutlich reduziert werden. Dabei ist der erforderliche finanzielle Aufwand verglichen mit traditionellen massiv parallelen Verfahren verhältnismäßig gering, so dass eine Gesamtlösung auch für kleine und mittelständische Unternehmen zugänglich ist. Gleichzeitig bieten GPUs ein günstigeres Verhältnis von Rechenleistung zu Energieverbrauch.

Abstract

Traditionally, computational fluid dynamics (CFD) is done in a cyclic sequence of independent steps. The simulation is usually set up in a pre-processing phase by defining the geometry of the flow domain as well as the initial and boundary conditions. After that the calculation is carried out for a fixed number of time steps. As CFD simulations are computationally intensive they are usually executed on high performance systems. In high performance computing (HPC) environments the job is typically submitted to a queueing system and is not executed until the required resources become available. The simulation may run for hours, days or even longer until eventually the results are stored on a file system. For the concluding post-processing these data are frequently transferred to the user's workstation to evaluate the simulation and identify errors or possible modifications to model parameters that will be applied in a next iteration. For each adaptation that is to be made to the computation, the whole process has to be repeated. User interaction is obviously rather limited and takes place exclusively during the pre-processing/modification phase. Not surprisingly, it is a long term wish of scientists and engineers to closely interact with their running simulations. Since the influential report of the US National Science Foundation (NSF) in 1987 new forms of scientific visualization have evolved that are quite different from traditional post-processing. Especially the approach commonly referred to as computational steering has been the subject of widespread interest in the scientific CFD community. Although it is a very powerful paradigm, the use of computational steering is still the exception rather than the rule. The reasons for this are more or less related to the complexity and restrictions of traditional HPC systems. As an alternative to the traditional massively parallel approach, in this thesis the parallel computational power of GPGPUs (General Purpose Graphics Processing Unit) is used for general purpose applications. The so called GPGPU computing has gained large popularity in the CFD community, especially for its application to the lattice Boltzmann method (LBM). Using this technology this work demonstrates a single desktop application integrating a complete interactive CFD simulation environment for reasonable hardware costs. It shows that the convergence of massive parallel computational power and steering environment into a single system significantly improves the usability, application quality and user-friendliness. Using multiple GPUs, the efficiency of this approach allows for CFD simulations in three dimensional space evolving close to real-time even for reasonable grid sizes. Thereby, the simulation can be explored and also adjusted during runtime. The thesis also shows that the responsiveness significantly benefits from avoiding common bandwidth and latency bottlenecks inherent in traditional HPC approaches. Those can be avoided as GPGPU computing does not generally require network communication, which also reduces the complexity of the application. Compared to traditional massive parallel environments, GPUs are affordable also by small to medium enterprises and do not require additional HPC knowledge from endusers. Besides, GPUs reduce power consumption by an improved performance-per-watt ratio.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ALU	Arithmetic Logic Unit
API	Application Programming Interface
AVO	Abstract Visualization Object
AVS	Advanced Visual Systems
BGK	Bhatnagar, Gross und Krook
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
CFX	steht für eine kommerzielle Simulationssoftware
CLB	Cascaded Lattice Boltzmann
COVISE	Collaborative Visualization Environment
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DIP	Dependency Inversion Principle
DMA	Direct Memory Access
DNS	Direct Numerical Simulation
DRAM	Dynamic Random-Access Memory
DRY	Don´t Repeat Yourself
FCoI	Favour Composition over Inheritance
FLOPS	Floating Point Operations per Second
FSI	Fluid-Struktur-Interaktion
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
GUI	Graphical User Interface
GUI	Heating, Ventilating and Air Conditioning
HDF	Hierarchical Data Format
HPC	High Performance Computing
IBM	International Business Machines
IRIS	Interior Redesign Industry Specialists
iRMB	Institut für rechnergestützte Modellierung im Bauingenieurwesen
ISP	Interface Segregation Principle
KISS	Keep it simple, stupid
LBM	Lattice-Boltzmann-Methode
LES	Large Eddy Simulation
LoD	Law of Demeter
LSP	Liskov Substitution Principle
MRT	Multiple-Relaxation-Time

MVC	Model View Controller
MVE	Modular Visualization Environments
MVP	Model View Presenter
NCSA	National Center for Supercomputing Applications
NSF	National Science Foundation
NUPS	Nodal Updates per Second
OCP	Open Closed Principle
OO	Object-Orientation
OOA	Object-oriented Analysis
OOD	Object-oriented Design
OOE	Out-of-Order Execution
PC	Personal Computer
PDA	Personal Digital Assistant
PSE	Problem Solving Environments
SAGA	Simple API for Grid Applications
SIMD	Single Instruction, Multiple Data
SM	Streaming-Multiprozessor
SMT	Simultaneous Multithreading
SOAP	Simple Object Access Protocol
SoC	Separation of Concerns
SP	Streaming-Prozessor
SRP	Single Responsibility Principle
STL	Surface Tesselation Language
VR	Virtual Reality
VTK	The Visualization Toolkit

1 Einleitung

1.1 Motivation

Vor über einem halben Jahrhundert proklamierte John von Neumann 1946 seine Idee von einem »Digitalen Windkanal« [200]. Motiviert durch die nachhaltige Stagnation in der Entwicklung analytischer Methoden zur Lösung partieller Differentialgleichungen, insbesondere im Bereich der Strömungsmechanik, sah von Neumann eine Alternative darin, einen neuartigen Digitalrechner in Kombination mit numerischen Verfahren zur Lösung komplexer Strömungsprobleme einzusetzen.

Durch die unvergleichliche Evolution der Leistungsfähigkeit von Rechnerarchitekturen und Algorithmen nehmen numerische Simulationen heute speziell im Bereich der Strömungsmechanik einen festen Platz neben Theorie und experimentellen Modellversuchen ein. Dabei haben sich Simulationen insbesondere in Bereichen, in denen Experimente unmöglich oder zu kostenintensiv sind bzw. Menschen und Umwelt gefährden würden, zu einem unverzichtbaren Instrument für Wissenschaft und Industrie entwickelt [133]. So erlauben es Simulationen, Strömungsphänomene geschlossener Systeme zu studieren, in denen keine Messungen durchgeführt werden können. Ein Beispiel dafür ist das Strömungsverhalten im Inneren des Zylinders eines Verbrennungsmotors. Gleichzeitig können Simulationsmodelle dafür herangezogen werden, zukünftige Situationen zu prognostizieren, die derzeit nicht messbar sind. Man denke an die Vorhersage des Wetters oder auch die Ausbreitung von Flutwellen bei Großschadensereignissen. Ein virtueller Versuch kann auch dann eine Alternative darstellen, wenn ein Laborexperiment mit exorbitanten Kosten verbunden wäre wie zum Beispiel der simulierte Start eines *Space-Shuttles*. Gleichzeitig können Parameteränderungen an Modellparametern in der Simulation sehr viel schneller und kostengünstiger als im realen Experiment durchgeführt werden und tragen damit zu einer erhöhten Produktivität bei.

Auf Grund der im Allgemeinen notwendigen hohen Rechenleistung wird die Mehrzahl der Strömungssimulationen heutzutage auf konventionellen Hochleistungsrechnern durchgeführt, wodurch der potentielle Anwenderkreis vor dem Hintergrund hoher finanzieller Kosten und erforderlicher interdisziplinärer Kompetenzen eingeschränkt wird. Eine zusätzliche Erschwernis stellen aus Benutzersicht die Interaktionsmechanismen und Analysemöglichkeiten der Simulationsumgebungen dar. Die notwendige hohe Rechenleistung wird jedoch stetig kostengünstiger und einfacher verfügbar. Allen voran ist die derzeitige Entwicklung auf dem Gebiet moderner Grafikprozessoren sehr vielversprechend, da ihre Rechenleistung auch für numerische Probleme herangezogen werden kann. So ermöglicht der Einsatz dieser Prozessoren die Entwicklung interaktiver Simulationssysteme, die kostengünstig sind und sich hinsichtlich ihrer Bedienung an konventionellen Desktop-Anwendungen orientieren, wodurch sich die Systeme einer potentiell größeren Anwendergruppe erschließen. Ausgehend von dieser Motivation wird im Rahmen der vorliegenden Arbeit eine interaktive Simulationsumgebung zur numerischen Berechnung von Strömungen auf Basis der Lattice-Boltzmann-Methode [149] vorgestellt.

1.2 Gliederung

Zu Beginn gibt Kapitel 2 eine Einführung in die Thematik der interaktiven Simulation, auch bezeichnet als *Computational Steering*. Dazu werden zunächst traditionelle Ansätze zur Lösung rechenintensiver Problemstellungen betrachtet und anschließend interaktiven Verfahren gegenübergestellt. Zu diesem Zweck wird die Historie des *Computational Steerings* erläutert und Vorteile des Paradigmas hervorgehoben. Als integraler Bestandteil des Analyseprozesses wird zudem die Rolle der wissenschaftlichen Visualisierung erläutert und abschließend der derzeitige Stand der Forschung betrachtet.

Im Mittelpunkt dieser Arbeit steht die Entwicklung der interaktiven Simulationsumgebung zur Berechnung von Strömungen VIRTUALFLUIDS INTERACTIVE. Zur Einordnung und Förderung des Verständnisses der nachfolgenden Kapitel wird die Anwendung im Rahmen von Kapitel 3 vorgestellt. Dazu erfolgt zunächst eine kurze Erläuterung von Funktion und Aufbau, die gleichzeitig die wichtigsten Aspekte dieser Arbeit hervorhebt.

Im Rahmen von Kapitel 4 wird die Basisarchitektur von VIRTUALFLUIDS INTERACTIVE vorgestellt. Dazu erfolgt zunächst eine Betrachtung der Aspekte eines qualitativ hochwertigen Softwareentwurfs.

Im Fokus von Kapitel 5 steht die Lattice-Boltzmann-Methode als numerisches Verfahren zur Simulation von Strömungen und seine effiziente Implementierung zum Einsatz auf Grafikkarten. Dazu erfolgt zunächst eine Erläuterung der Grundlagen des numerischen Verfahrens, sowie der architekturellen Besonderheiten von GPUs und ihrer Programmierung.

In Kapitel 6 wird die Bedeutung der Visualisierung für die Analyse von Ergebnisdaten verdeutlicht. Es wird ein Überblick über die in VIRTUALFLUIDS INTERACTIVE verwendeten Verfahren gegeben und ein komponentenbasierter Ansatz vorgestellt, der den Einsatz von Visualisierungstechniken im Rahmen eigener Erweiterungen erleichtert. Zudem werden Limitierungen klassischer Verfahren und Systeme im Zusammenhang mit der Verarbeitung CFD-typischer Massendaten diskutiert und Lösungsansätze vorgestellt.

Abschließend werden in Kapitel 7 einige ausgewählte Anwendungsbeispiele aus dem Gebiet des Bauingenieurwesens vorgestellt. Diese zeigen auf, dass die computergestützte Simulation im Bereich der Strömungsmechanik generell als wertvolles Hilfsmittel für Ingenieure und Wissenschaftler dienen kann. Vielmehr wird jedoch darauf aufmerksam gemacht, dass sich die notwendigen Werkzeuge dank der Leistung moderner Grafikprozessoren in die gewohnte Arbeitsumgebung integrieren lassen, ohne bedeutende finanzielle Investitionen oder interdisziplinäre Kompetenzen zu erfordern.

2 Grundlagen

Dieses Kapitel dient als Einführung in die Thematik der interaktiven Simulation, auch bezeichnet als *Computational Steering*. Dazu werden zunächst traditionelle Ansätze zur Lösung rechenintensiver Problemstellungen betrachtet und anschließend interaktiven Verfahren gegenübergestellt. Zu diesem Zweck werden die Historie des *Computational Steerings* erläutert und Vorteile des Paradigmas hervorgehoben. Als integraler Bestandteil des Analyseprozesses wird zudem die Rolle der wissenschaftlichen Visualisierung erläutert und abschließend der derzeitige Stand der Forschung betrachtet.

2.1 Traditionelle Simulationsverfahren

Traditionell werden rechenintensive Simulationen in einer zyklischen Sequenz autonomer Teilschritte durchgeführt. Dabei sind die einzelnen Teilschritte oft mit hohem zeitlichen, personellen und rechnerischen Aufwand verbunden. Diese Aussage ist insbesondere für Simulationen raum-zeitlicher physikalischer Prozesse wie beispielsweise numerische Strömungssimulationen (CFD) (*Computational Fluid Dynamics*) gültig.

2.1.1 Pre-Processing

Am Anfang der Prozesskette steht in der Regel das sogenannte *Pre-Processing*[1]. In diesem Verarbeitungsschritt werden die Eingangsdaten für die nachfolgende Simulation vorbereitet. Den Ausgangspunkt stellt im Regelfall ein geometrisches Modell dar, das im Falle einer CFD-Simulation die Geometrie des Strömungsgebiets beschreibt. Diese Beschreibung muss mit Rand- und Anfangsbedingungen versehen und zunächst in eine diskrete Form überführt werden. Die diskrete Form wird üblicherweise durch ein Berechnungsgitter dargestellt, das abhängig von dem verwendeten Verfahren unterschiedliche Strukturen aufweisen kann. Abbildung 2.1 stellt einige gängige Gitterstrukturen dar. Der Prozess der Gittergenerierung kann abhängig von der Geometrie und der gewählten Gitterstruktur überaus komplex und rechnenintensiv sein und damit einen Großteil der *Pre-Processing*-Phase ausmachen. Dieser Rechenaufwand ist insbesondere in Hinblick auf adaptive Verfahren von Bedeutung, bei denen eine Anpassung der Diskretisierung zur Laufzeit erfolgt. Das *Pre-Processing* hat zudem einen besonderen Stellenwert, da die Qualität des Berechnungsgitters einen entscheidenden Einfluss auf die Genauigkeit und Qualität der Simulationsergebnisse hat.

2.1.2 Simulation

Der anschließende Simulationsschritt stellt hinsichtlich des Berechnungsaufwands eine besondere Herausforderung dar. Seit jeher besteht eine enge Beziehung zwischen Anwendungen des wissenschaftlichen Rechnens (insbesondere des CFD-Bereichs) und dem Hochleistungsrechnen (HPC)

[1] Vorverarbeitung

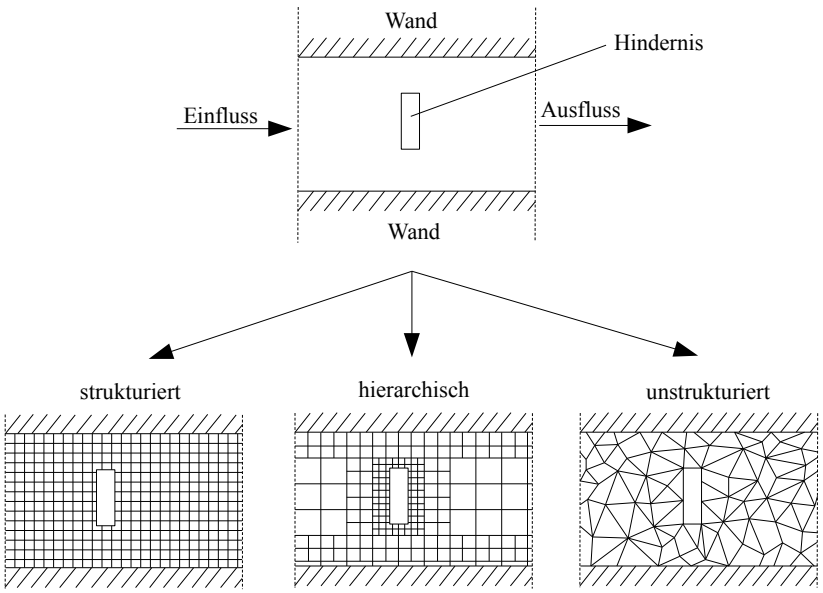


Abbildung 2.1: *Diskretes Modell auf Basis unterschiedlicher Gittertypen.*

(*High Performance Computing*), da wissenschaftliche Simulationen auf Grund stetig wachsender Problemgröße und Komplexität erhebliche Anforderungen an die Rechenkapazität stellen. Um diesen Anforderungen nachzukommen, werden die Simulationen im Allgemeinen von Hochleistungsrechnern wie beispielsweise traditionellen Supercomputern [212] oder Computerclustern [51] durchgeführt (vgl. Abschnitt 5.2). Die Anschaffung und Nutzung derartiger Systeme ist üblicherweise mit immensen Kosten verbunden und damit in der Regel nur einem eingeschränkten Anwenderkreis zugänglich. Zur optimalen Nutzung der Systeme wird die Ressourcenverwaltung üblicherweise automatisiert. Die Durchführung der Berechnungen erfolgt dabei allgemein in Form einer Stapelverarbeitung (*Batch Processing*). Dazu wird die Simulation an ein *Queueing System* (Warteschlangensystem) übergeben, das die Berechnungen ausführt, sobald die notwendigen Ressourcen verfügbar sind. Die Laufzeit der nun folgenden Simulationsphase basiert im Wesentlichen auf einer im Vorfeld definierten Anzahl an Zeitschritten und kann leicht mehrere Stunden, Tage oder länger andauern. Schlussendlich liegen die Ergebnisdaten im Dateisystem vor.

2.1.3 Post-Processing

Der abschließende *Post-Processing*-Schritt[2] der Berechnungsergebnisse erfolgt zumeist nicht (lokal) auf dem Hochleistungsrechner. Stattdessen werden die Ergebnisdaten auf ein geeignetes System transferiert (verteilt), wodurch abhängig vom Datenvolumen zusätzliche Zeit in Anspruch genommen wird. In der Regel sind heutige *Workstations*[3] hinreichend leistungsfähig, so dass die Analyse-

[2] Nachbearbeitung, Analyse
[3] Arbeitsplatzrechner

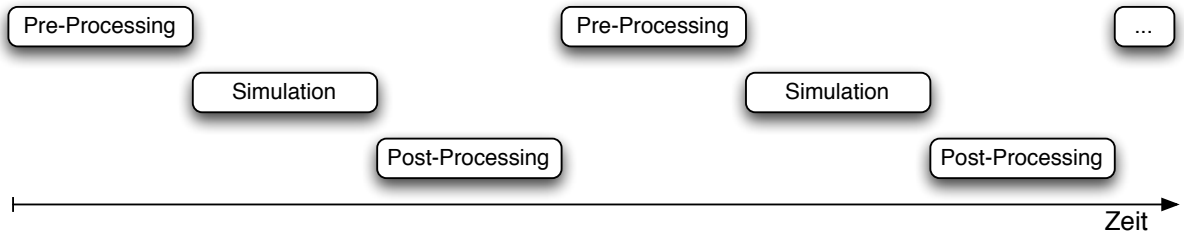


Abbildung 2.2: *Darstellung der traditionellen HPC-Prozesskette basierend auf [300].*

und Interpretationsphase direkt am Arbeitsplatz des jeweiligen Anwenders erfolgt [81, 189]. Einzig und allein in diesem Teilschritt ist es den Ingenieuren und Wissenschaftlern möglich, die anfallenden Daten der Simulation auszuwerten, indem diese beispielsweise zu Zwecken der Validierung mit Ergebnissen aus Experimenten verglichen werden [214]. Aus den identifizierten Abweichungen lassen sich notwendige Änderungen für die nächste Iteration ableiten. Damit diese Anpassungen Anwendung finden, muss jedoch die gesamte Prozesskette (vgl. Abbildung 2.2) wiederholt werden, da Änderungen an den Parametern bei konventionellen Verfahren ausschließlich in der Phase des *Pre-Processings* bzw. zu Beginn der Simulation erfolgen können. Ingenieure und Wissenschaftler fordern daher Möglichkeiten zur Benutzerinteraktion mit der Simulation.

2.2 Computational Steering

Erstmals formuliert wird der Wunsch nach mehr Interaktion mit laufenden Simulationen bereits im Jahr 1987 in einem bedeutenden Bericht der *National Science Foundation* (NSF) [188]. Dort heißt es:

> »Scientists not only want to analyze data that results from super-computations; they also want to interpret what is happening to the data during super-computations. Researchers want to steer calculations in close-to-real-time; they want to be able to change parameters, resolution or representation, and see the effects. They want to drive the scientific discovery process; they want to interact with their data.«[4]

Im Jahr darauf verfasst Brooks einen Artikel, in dem er die Notwendigkeit von generischen Anwendungen zur Steuerung großer Simulationsläufe zum Ausruck bringt [40]. Die Realisierbarkeit der innovativen Konzepte wurde erstmalig von Haber and McNabb auf der SIGGRAPH-Konferenz im Jahr 1989 eindrucksvoll demonstriert [112]. Dabei wurde eine auf einem Supercomputer im US-Bundesstaat Illinois ausgeführte Simulation von einem Anwender auf der Konferenz in Boston interaktiv gesteuert. Ein frühes Beispiel einer tatsächlichen Anwendung wird von Marshall et al. beschrieben [176]. Sie stellen ein System zur interaktiven Steuerung und Analyse eines Turbulenzmodells am Beispiel des Eriesees vor. Marshall et al. subsumieren die interaktiven Eigenschaften des Systems unter dem Begriff *Computational Steering*. In den Folgejahren leisten unter anderem

[4] »Wissenschaftler wollen nicht nur die Ergebnisdaten ihrer Supercomputer-Berechnungen analysieren; sie möchten auch interpretieren können, was mit diesen Daten während der Berechnungen geschieht. Forscher möchten ihre Berechnungen in nahezu Echtzeit steuern; sie wollen in der Lage sein, Änderungen an Parametern, Auflösung oder Repräsentation vorzunehmen und die Auswirkungen zu beobachten. Sie wollen den wissenschaftlichen Erkundungsprozess lenken; sie möchten mit ihren Daten interagieren.«

van Liere, Mulder und van Wijk [163, 164, 165, 195, 196, 304, 305] ebenso wie Johnson und Parker [133, 134, 214, 215, 216, 217] einen erheblichen Beitrag zur Entwicklung des *Computational Steering*-Paradigmas.

2.2.1 Definition

Marshal et al. unterscheiden zwischen zwei Alternativen zum traditionellen *Post-Processing*-Ansatz. Ein erster Schritt zu mehr Interaktion mit laufenden Simulationen basiert auf dem sogenannten *Tracking* [176], das von Hart und Kraemer auch als *Monitoring* [118] bezeichnet wird. Dabei werden nach einer definierten Anzahl an Zeitschritten Zwischenergebnisse an den Analyseprozess übermittelt. Auf diese Weise ist es bereits in einem frühen Stadium möglich, die Gültigkeit der Ergebnisse zu überprüfen. Auf dieser Grundlage kann gegebenenfalls entschieden werden, die Simulation zu stoppen und anschließend mit veränderten Parametern zu wiederholen. Demgegenüber geht *Computational Steering* einen erheblichen Schritt weiter. Mulder und andere definieren es als die Fähigkeit, alle Aspekte eines Simulationsprozesses bestehend aus *Pre-Processing*, Simulation und *Post-Processing* während ihrer Ausführung interaktiv kontrollieren zu können [195]. Nach ihrer Aussage kann *Computational Steering* als ultimatives Ziel des interaktiven Rechnens verstanden werden [163, 165, 196, 305]. Als solches bricht *Computational Steering* den traditionellen Simulationszyklus auf, indem es Ingenieuren und Wissenschaftlern ermöglicht, unmittelbar auf Ergebnisse zu reagieren und Simulationsparameter zur Laufzeit zu verändern. Der Anwender wird auf diese Weise integraler Bestandteil des Simulationsprozesses. Vielfach wird daher auch die Bezeichnung *Human in the Loop* verwendet [292].

Anwendungen wie das von Marshal et al. beschiebene System zur interaktiven Simulation, werden allgemein auch als *Computational Steering Environment* (CSE) bezeichnet [81, 165, 195, 215]. Umfasst das System alle notwendigen Komponenten zur Lösung von Problemen einer bestimmten Anwendungsdomäne, wird häufig auch die Bezeichnung *Problem Solving Environment* (PSE) verwendet [126]. Es sei an dieser Stelle angemerkt, dass diese Begriffe in der Literatur zumeist nicht einheitlich definiert werden. Unterschiedliche Forschungsgruppen favorisieren zudem bestimmte Begrifflichkeiten und verwenden korrespondierende Ausdrücke wiederum selten. In der vorliegenden Arbeit wird vorzugsweise der Begriff *Computational Steering* verwendet. Analog dazu wird die im Rahmen dieser Arbeit entwickelte Anwendung als *Computational Steering*-Umgebung oder auch als interaktive Umgebung bezeichnet.

2.2.2 Anwendungsfelder

Computational Steering eignet sich zur Anwendung in vielen Bereichen der Wissenschaft. Im ingenieurwissenschaftlichen Umfeld steht der Begriff dabei in engem Bezug zu Simulationsprozessen. In diesem Bereich unterscheiden Mulder et al. drei wesentliche Anwendungsfelder [195]:

- Exploration des Modells
- Algorithmische Untersuchungen
- Performanceoptimierung

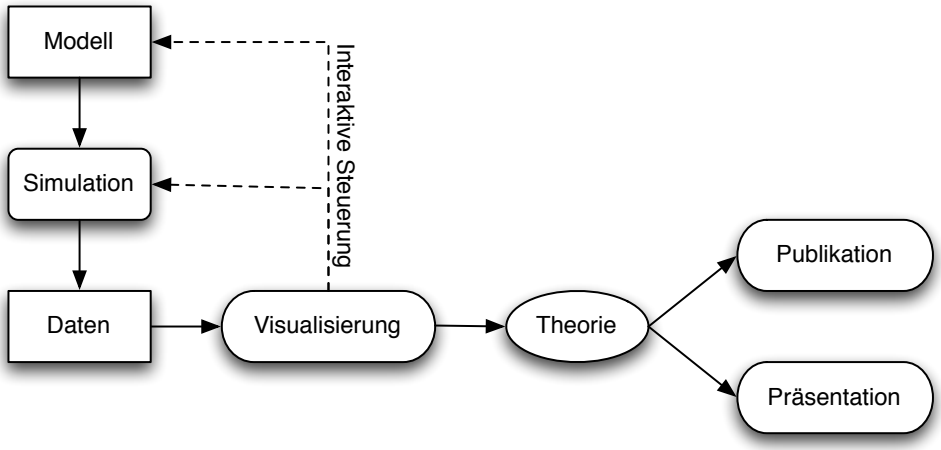


Abbildung 2.3: *Computational Steering als Mittel zur Exploration des Modells nach [293].*

Unter der Exploration des Modells mit Hilfe von *Computational Steering* wird die Erkundung des Parameterraums und des Verhaltens der Simulation auf Variationen einzelner Parameter subsumiert. Dieser Aspekt ist in der vorliegenden Arbeit von besonderer Bedeutung. Die Untersuchung von Algorithmen kann beispielsweise Aufschluss über verschiedene Lösungsverfahren liefern, indem diese miteinander verglichen werden. Dazu kann der Anwender Algorithmen zur Laufzeit variieren. Im Rahmen einer Performanceoptimierung wird der Benutzer in die Lage versetzt, das Laufzeitverhalten der Simulation zu beeinflussen, indem dieser beispielsweise eine interaktive Lastbalancierung einer parallel ausgeführten Berechnung vornimmt.

2.2.3 Vorteile

Allgemein bietet *Computational Steering* insbesondere bei der Berechnung transienter Phänomene gegenüber traditionellem *Post-Processing* mehrere wesentliche Vorteile [55, 153, 176, 195, 214, 215, 216, 222, 304]. Ein entscheidendes Merkmal sind kürzere Iterationszyklen und eine damit einhergehende Reduktion der Gesamtlaufzeit, die letztlich eine Produktivitätssteigerung mit sich bringt. Erreicht wird dies zum einen durch das *Monitoring* laufender Simulationen, mit deren Hilfe beispielsweise Fehler frühzeitig identifiziert werden können und somit redundante Berechnungen vermieden werden. Eine erhebliche zusätzliche Steigerung der Produktivität wird zum anderen durch kombinierte Interaktionsmöglichkeiten erreicht, die den Benutzer in die Lage versetzen, unmittelbar auf aktuelle Simulationsergebnisse zu reagieren und somit direkten Einfluss auf den Simulationsverlauf zu nehmen. Gleichzeitig spiegeln sich Änderungen an Simulationsparametern instantan im *Monitoring* wider, so dass die Beziehung zwischen Ursache und Wirkung ersichtlich wird und der Anwender eine Intuition hinsichtlich der Auswirkungen von Parameteränderungen entwickeln kann. Auf diese Weise wird es den Ingenieuren und Wissenschaftlern ermöglicht, Fehler aufzudecken und ein tieferes Verständnis der physikalischen Zusammenhänge und Algorithmen zu erlangen. Der wissenschaftliche Erkundungsprozess basiert zu einem erheblichen Teil auf der Beantwortung der Frage »Was wäre wenn?« (*what-if-analysis* [103]). Dieser Frage kann mit Hilfe von *Computational Steering* deutlich effizienter nachgegangen werden. Innerhalb einer Gruppe stellt es somit ein hilfreiches Mittel zur Kommunikation und Diskussion dar, indem es das Modell sowie dessen Ergebnisse repräsentiert und

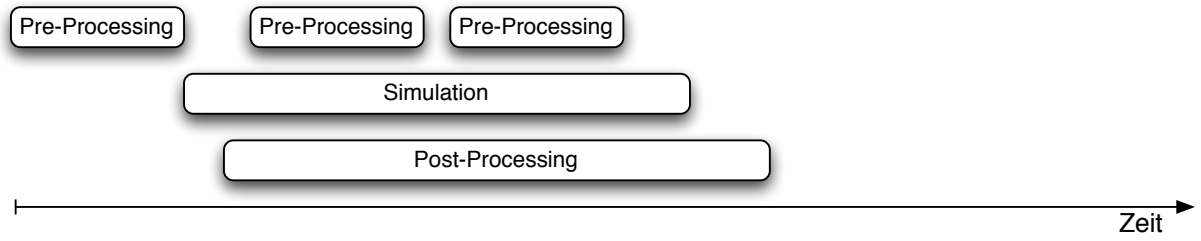


Abbildung 2.4: *Darstellung der optimierten Prozesskette des Computational Steering Ansatzes basierend auf [300].*

umgehend Antworten auf Fragen liefert. Vor diesem Hintergrund eignet sich hochinteraktives *Computational Steering* insbesondere für das *Rapid Prototyping*[5]. Im Fall langer Produktionsläufe mit umfangreichen Problemgrößen, für die ein geeignetes Setup und Simulationsparameter bestimmt sind, ist der Einsatz von *Computational Steering* gegebenenfalls weniger vorteilhaft [68, 153, 216].

Abbildung 2.4 zeigt schematisch die deutlich verkürzte Prozesskette des *Computational Steering*-Verfahrens.

2.3 Erkenntnis durch Visualisierung

Für bestimmte wissenschaftliche Problemstellungen ist es möglich, eine automatisierte Analyse und Interpretation der Ergebnisdaten vorzunehmen. Auf dem Gebiet der Strömungssimulation sind Wissenschaftler und Ingenieure unter anderem daran interessiert zu erfahren, ob ihre numerischen Verfahren konvergieren und ob die Massenerhaltung gegeben ist. Diese Informationen lassen sich mit geringem Aufwand berechnen und auf einzelne Ergebniswerte reduzieren. Häufig ist eine automatisierte Analyse jedoch nicht möglich, weil keine Techniken existieren, um ein bestimmtes Merkmal zu extrahieren oder weil nichts über den Informationsgehalt der vorliegenden Daten bekannt ist. In diesem Fall besteht die einzige Alternative häufig darin, eine manuelle Analyse vorzunehmen.

2.3.1 Abgrenzung wissenschaftlicher Visualisierung

Während Ingenieure und Wissenschaftler in der Anfangszeit des wissenschaftlichen Rechnens noch gezwungen waren, Berechnungsergebnisse anhand von Ausdrucken der Rohdaten auszuwerten [214], stellt heutzutage die Visualisierung einen integralen Bestandteil des Analyseprozesses dar. Diese Aussage gilt gleichermaßen für traditionelle wie auch für interaktive Simulationsverfahren. Es ist allgemein unbestritten, dass die Visualisierung eine Schlüsselfunktion zum Verständnis komplexer abstrakter Informationen und Modelle einnimmt, da sie sich die enorme Leistungsfähigkeit des menschlichen visuellen Cortex zunutze macht [68, 172, 197]. Auf diese Weise ist es dem Menschen möglich, Informationen weitaus effizienter aufzunehmen, als es ihm durch Studieren der Rohdaten möglich wäre. Dabei besteht die herausfordernde Aufgabe der Visualisierung darin, komplexe häufig multi-dimensionale Daten so in eine expressive grafische Repräsentation

[5] Rapid Prototyping (schneller Prototypenbau) kann hier verstanden werden als eine Methodik zur Erstellung eines funktionsfähigen Prototypen, um die Praktikabilität eines Verfahrens bzw. Modells zu demonstrieren.

umzuwandeln, dass die relevanten Eigenschaften intuitiv erfasst, verstanden und bewertet werden können und zusätzlich die Integrität der Informationen gewährleistet ist [188, 240]. Handelt es sich bei den zu betrachtenden Daten um Ergebnisse aus Simulationen, Experimenten oder allgemein wissenschaftlichen Berechnungen, so wird auch von wissenschaftlicher Visualisierung (*Scientific Visualization*) [38, 188, 202, 313] gesprochen. Dieser Begriff ist ebenfalls durch den Bericht der NSF [188] des Jahres 1987 geprägt. Angetrieben durch das Gesetz von Moore [194] fallen insbesondere im Bereich wissenschaftlicher Simulationen auf Grund stetig wachsender Rechenleistung immer komplexere und größere Datenvolumina an. Die Aufbereitung dieser Daten stellt für die Ingenieure und Wissenschaftler eine immense Herausforderung dar. Von besonderer Bedeutung ist dabei die Visualisierung als unverzichtbares Hilfsmittel, das es bei korrekter Anwendung ermöglicht, die Fülle an Informationen zu untersuchen und bestimmte Muster, Strukturen sowie Anomalien zu erkennen und dabei interne, sonst verborgene Zusammenhänge aufzudecken [53, 68, 172]. Entscheidend für die Erkenntnis sind die menschlichen Fähigkeiten der visuellen Wahrnehmung und Kognition sowie die Einbeziehung von a-priori-Wissen bezüglich des betrachteten Phänomens [17, 197, 306]. Die Kombination dieser Fähigkeiten ermöglicht dem Analysten das Bilden eines mentalen Modells[6] auf Basis der zugrundeliegenden Daten und fördert dadurch das Verständnis [247].

2.3.2 Erfolgreiche Visualisierung durch Interaktion

Generell ist die wissenschaftliche Visualisierung ein iteratives Verfahren [172, 305], bei dem relevante Information aus den Daten in der Regel erst durch eine Variation der Visualisierungsmethoden und ihrer Parameter zum Vorschein gebracht werden. Daher ermöglicht ein gutes Visualisierungssystem dem Anwender, den Parameterraum auf intuitive Weise interaktiv zu erkunden. Je effizienter dabei das System ist, desto weniger Iterationen in Form von Parameteränderungen sind für eine erfolgreiche Exploration erforderlich [172]. Diese Form der Analyse wird auch als interaktive Visualisierung bezeichnet [55]. In diesem Zusammenhang ist zu beachten, dass sich die Interaktivität zunächst ausschließlich auf die Visualisierung bezieht und nicht gleichbedeutend mit der Interaktion im Kontext laufender Simulationen ist. Als solche wird die interaktive Visualisierung als essentielles Hilfsmittel sowohl für die visuelle Aufbereitung bereits existierender Daten, als auch für das *Computational Steering* eingesetzt. Im Rahmen der Analyse laufender Strömungssimulationen ermöglicht der Einsatz einer interaktiven Visualisierung beispielsweise Zwischenergebnisse, Parameter und Randbedingungen zu kontrollieren und die Berechnungen auf Grund dieser Erkenntnisse entsprechend anzupassen.

Im Allgemeinen zählen heute zahlreiche Bereiche der Wissenschaft zu den Anwendungsgebieten der Visualisierung. Sie wird unter anderem in den Ingenieurwissenschaften, der Chemie, Medizin, Physik und Mathematik erfolgreich eingesetzt. Zu den klassischen Bereichen der Visualisierung zählen insbesondere die Humanmedizin (vgl. Abbildung 2.5) sowie die Visualisierung von Strömungen (vgl. Abbildung 2.6), die insbesondere für diese Arbeit von Bedeutung ist.

[6] Der Begriff mentales Modell oder Vorstellungsbild entstammt der kognitiven Psychologie und bezeichnet dort eine Form der Wissensrepräsentation beim Menschen [7].

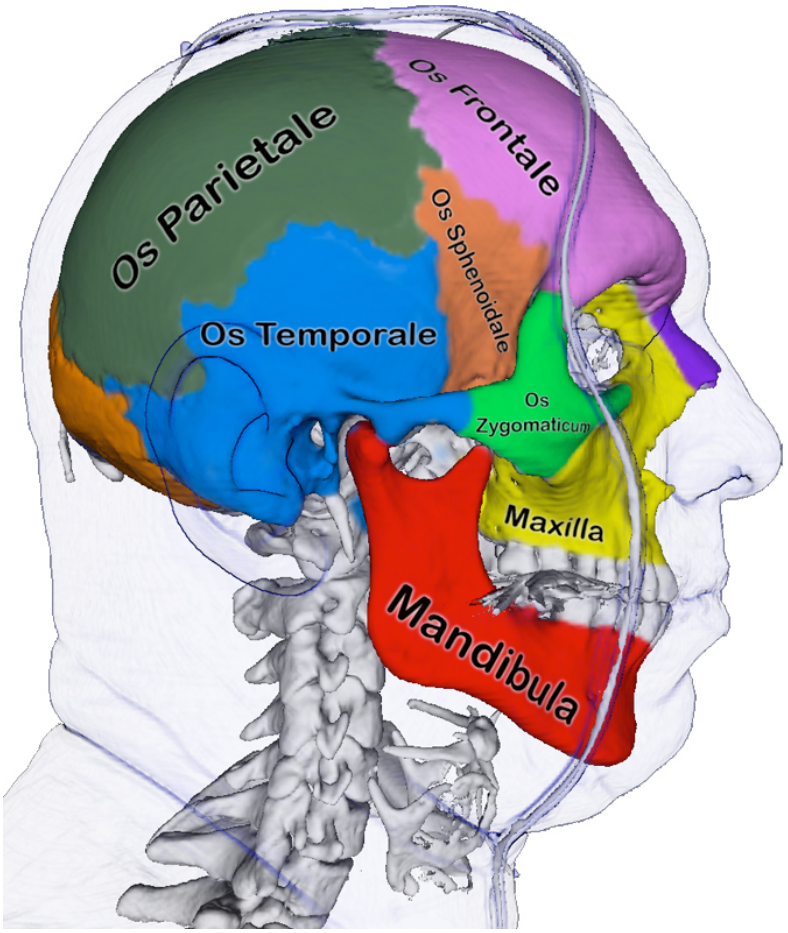


Abbildung 2.5: *Volumendarstellung der menschlichen Schädelknochen [248].*

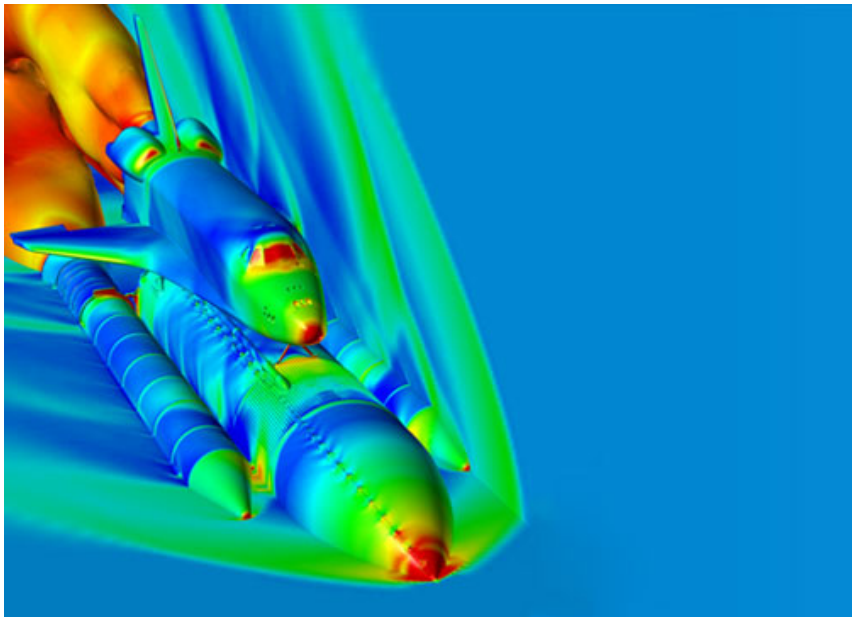


Abbildung 2.6: *Druckverteilung im Strömungsfeld auf ein Space Shuttle [198].*

2.4 Status Quo - Stand der Technik

Obwohl die Vorteile von *Computational Steering* offensichtlich sind, ist seine Anwendung, insbesondere auf dem Gebiet der numerischen Strömungssimulation, aber auch in anderen Bereichen des Ingenieurwesens und der Naturwissenschaften, heutzutage noch immer eine Ausnahme [222, 301, 315]. Der Grund dafür sind verschiedene Umstände, die häufig in Bezug zum Hochleistungsrechnen stehen. Die komplexe Aufgabe, eine *Computational Steering*-Anwendung zu entwickeln, erfordert die Integration zahlreicher unterschiedlicher Disziplinen [32, 217, 304]. Bei Anwendung auf die Strömungsmechanik sind beispielsweise Kenntnisse aus den Bereichen Physik, Numerik, Softwareentwicklung, Design grafischer Benutzerschnittstellen (GUI) (*Graphical User Interface*), Computergrafik und Hochleistungsrechnen erforderlich. Erschwerend kommt hinzu, dass die Konfiguration von HPC-Umgebungen in der Regel keine interaktiven Berechnungen zulässt. Stattdessen wird die Ausführung der Berechnungen und somit die Verwendung der verfügbaren Ressourcen, wie bereits in Abschnitt 2.1.2 erwähnt, von Managementsystemen verwaltet. Diese Systeme basieren auf zahlreichen Annahmen, die mit *Computational Steering* nicht zu vereinbaren sind. Beispielsweise erfordert der interaktiver Betrieb, dass bestimmte Systemressourcen zu einem definierten Zeitpunkt für eine gewisse Dauer exklusiv zur Verfügung stehen. Daraus resultiert das verbreitete Vorurteil, Interaktivität verschwende Rechenzeit [133].

2.4.1 Entwicklungsschwerpunkte

Seit dem ausschlaggebenden Report der NSF konzentrierten sich die Entwicklungen zunächst auf Techniken der (interaktiven) Visualisierung, wohingegen nur wenige Arbeiten auf dem Gebiet der interaktiven Steuerung entstanden [165, 196, 300]. Dieser Verlauf ist insbesondere wegen der großen Bedeutung der Visualisierung für den Auswertungsprozess wenig erstaunlich. Auf diese Weise initiiert, begann die Entwicklung sogenannter modularer Visualisierungsumgebungen (MVE) (*Modular Visualization Environment*) [314]. Cameron gibt eine hilfreiche Zusammenfassung über die Merkmale einer solchen Umgebung [52]. Anders als anwendungsspezifische Umgebungen bieten diese die Möglichkeit, Daten beliebigen Ursprungs mit Hilfe generischer wissenschaftlicher Visualisierungsmodule darzustellen. Dabei bestimmt der Anwender den Datenfluss durch grafisches Verbinden einzelner Komponenten und kann auf diese Weise auch ohne Programmierkenntnisse eine Visualisierungsanwendung erstellen. Dieses sogenannte visuelle Programmierparadigma (*Visual Programming Paradigm*) wurde von Upson et al. mit AVS/Express [290], einer der ersten modularen Visualisierungsumgebungen, vorgestellt. Obgleich ein sehr mächtiges Werkzeug, erfordert dieses Prinzip vom Anwender jedoch intensive Kenntnis wissenschaftlicher Visualisierungskonzepte. Zur selben Zeit präsentierten Haber und McNabb eine elegante Abstraktion des Datenflusskonzeptes [111], dessen Grundidee darin besteht, dass Daten entlang einer Pipeline »fließen« und verarbeitet werden. Abbildung 2.7 zeigt eine schematische Darstellung des Datenflussmodells, dessen detaillierte Behandlung in Abschnitt 6.2.1 erfolgt.

Das Haber und McNabb Referenzmodell bildet noch heute die Basis vieler Entwicklungen auf dem Gebiet der Visualisierung und findet beispielsweise auch in späteren modularen Visualisierungsumgebungen wie dem IRIS Explorer [294] und dem IBM Data Explorer [2] (heute OpenDX) Anwendung. Ein bekannter Nachteil des Datenflusskonzeptes besteht in einem hohen Speicherbedarf, der sich

Abbildung 2.7: *Schematische Dartellung des Haber und McNabb Datenflussmodells [111].*

insbesondere limitierend auf die Verarbeitung CFD-typischer großer, zeitabhängiger Volumendaten auswirkt. Die Visualisierung derartiger Datenmengen stellt nach wie vor eine der größten Herausforderungen im Umfeld des wissenschaftlichen Rechnens dar und ist für diese Arbeit von maßgeblicher Bedeutung. Abschnitt 6.5 behandelt daher Strategien zur Vermeidung des erhöhten Speicherbedarfs. Aufgrund ihrer universellen Anwendbarkeit besitzen modulare Visualisierungsumgebungen meist einen sehr großen Funktionsumfang und damit eine hohe Komplexität. Jede Disziplin stellt jedoch unterschiedliche Anforderungen an die Visualisierung, so dass Anwender häufig spezifische Funktionalitäten vermissen, wohingegen entbehrliche Funktionen eine unnötige Komplexität bewirken.

2.4.2 Erweitertes Datenflussmodell

Spätere Generationen von modularen Visualisierungsumgebungen bieten die Besonderheit, die Datenflusspipeline um eigene Module erweitern zu können. Neben der Erweiterung um fachspezifische Visualisierungsfunktionen besteht eine Entwicklung darin, die Umgebungen zum *Monitoring* laufender Simulationen einzusetzen. Diese Möglichkeit findet damals wie heute vielfach Anwendung und stellt einen entscheidenden Schritt in Richtung *Computational Steering* dar. Frühe Anwendungsbeispiele hierfür sind CUMULVS [103] und VISIT [80]. Infolge der inhärenten Unidirektionalität der Datenflusspipeline sind Interaktionen mit der Simulation bei diesen Ansätzen meist nur rudimentär möglich. Häufig wird *Computational Steering* daher irrtümlich auf seine *Monitoring*-Fähigkeit reduziert [222, 312] und als Erweiterung der Pipeline verstanden, die Simulationsdaten auf der einen Seite empfängt und eine visuelle Darstellung auf der anderen Seite generiert, ohne dass eine intrinsische Verbindung zwischen beiden Enden besteht [311].

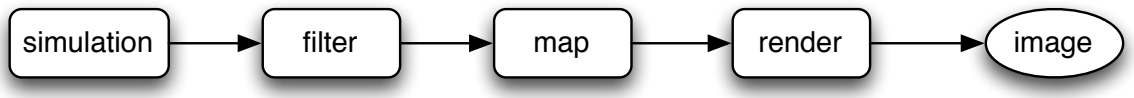


Abbildung 2.9: *Monitoring - ein erweitertes Datenflussmodell [314].*

2.4.3 Problem Solving Environments

Einen bedeutenden Schritt in Richtung Benutzerinteraktion gehen *Problem Solving Environments* (vgl. Abschnitt 2.2.1) wie beispielsweise SCIRun [217]. Sie adressieren vor allem die oft hinderliche Trennung von Analyse- und Simulationsumgebung, indem alle notwendigen Komponenten zur Problemlösung in einer einzelnen Anwendung konsolidiert werden [216]. So stellt SCIRun dem Anwen-

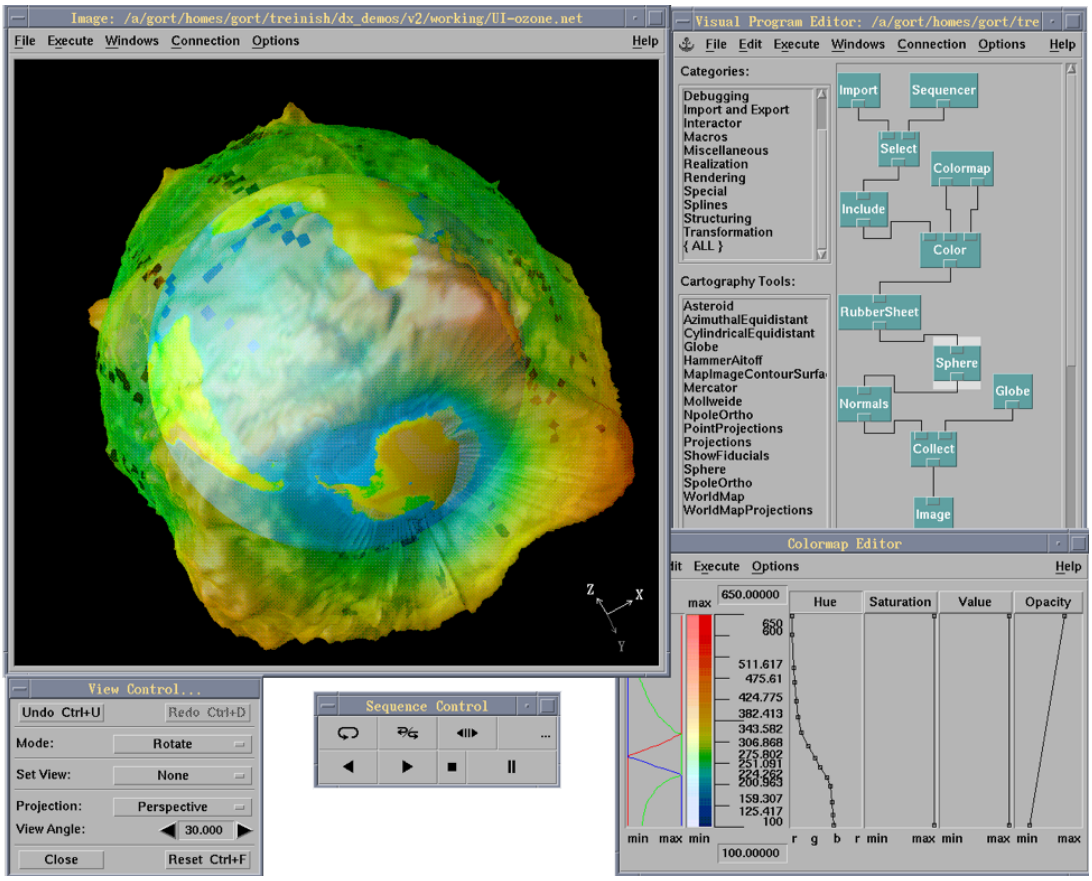


Abbildung 2.8: *Veranschaulichung des visuellen Programmierparadigmas anhand des frühen Beispiels der grafischen Benutzeroberfläche des IBM Data Explorers [2]. Die Darstellung zeigt eine 3D-Visualisierung der Ozonschicht der Erde [283].*

der eine universelle Plattform zur Verfügung, interaktive Simulationen zu entwickeln, zu testen und auszuführen. Dabei wird hinsichtlich der Benutzeroberfläche jedoch nicht zwischen Entwurf und Endanwendung unterschieden, wodurch auch Endanwender mit der vollständigen Komplexität der Plattform konfrontiert werden. Daher sind die Benutzergruppen der Entwickler und Endanwender in der Regel identisch.

2.4.4 Grid-Computing Initiative

Die Intention der *Problem Solving Environments*, alle Module einer Simulation auf einem einzelnen Rechner durchzuführen, führt eine deutliche Einschränkung hinsichtlich der Systemgröße mit sich, die insbesondere bei rechenintensiven CFD-Simulationen zum Tragen kommt. Mit Aufkommen des sogenannten Grid-Computings [90] wird daher vorzugsweise ein verteilter Ansatz verfolgt, indem die einzelnen Module als Services umgesetzt werden [36]. Auf diese Weise wird es möglich, die grafische Benutzeroberfläche zur Steuerung und Analyse der Simulation lokal auf einer Workstation auszuführen, während aufwändige Berechnungen, zu denen auch Teile der Visualisierung gehören können, in eine entfernte HPC-Umgebung ausgelagert werden. Serviceorientierte Grid-Umgebungen begünstigen zusätzlich die Entwicklung kollaborativer Anwendungen, die eine Zusammenarbeit von Experten an unterschiedlichen Orten ermöglichen [30, 150, 170]. Auf diese Weise wird es möglich, Kompetenzen unterschiedlicher Disziplinen virtuell zu bündeln und so der fortschreitenden Komplexität des Simulationsprozesses zu begegnen. Die Integration der Anwendungsmodule in eine Grid-Infrastruktur erfordert jedoch eine zusätzliche Middleware[7], mit deren Hilfe die Kommunikation einzelner Komponenten untereinander ermöglicht wird. Der Umsetzung einer solchen Infrastruktur sind zahlreiche Projekte gewidmet, aus denen RealityGrid [39, 222] und gViz [37, 309] besonders hervorstechen. Die Entwicklungen auf dem Gebiet der Grid-Infrastruktur schreiten derzeit rapide voran. Wright bemängelt jedoch das Fehlen von Standards und Stabilität auf Grund der schnellen Evolution [315]. Deshalb stellt die architekturelle Komplexität verteilter Anwendungen nach wie vor eine große Herausforderung dar. Die Kommunikation zwischen den einzelnen Modulen ist insbesondere im Kontext von *Computational Steering* von essentieller Bedeutung für die Qualität interaktiver Anwendungen. Limitierende Faktoren sind dabei in der Regel die Bandbreite sowie die Latenz von Netzwerkverbindungen.

2.5 Verwandte Arbeiten

Interaktive Techniken des *Computational Steerings* finden bereits seit Anfang der 1990er Jahre im Bereich numerischer Strömungssimulationen Anwendung. Dennoch haben die Konzepte bis dato kaum Einzug in kommerziell verfügbare Pakete zur Simulation von Strömungen gehalten. So erfolgt die Auswertung der Ergebnisse in Anwendungen wie CFX [9], Fluent [8] oder PowerFLOW [83] nach wie vor im *Post-Processing*. Dabei ist allgemein zu beobachten, dass die Visualisierungsfähigkeiten der

[7] Nach Tanenbaum [271] wird zwischen einem Computernetzwerk und einem verteilten System unterschieden. Im Unterschied zu einem Computernetzwerk stellt sich der Verbund aus unabhängigen Computern in einem verteilten System dem Anwender transparent als ein einzelnes kohärentes System dar. In der Regel repräsentiert es dabei ein einzelnes Modell oder Paradigma, das häufig von einer Software oberhalb des Betriebsystems implementiert wird. Diese Software wird auch als Middleware bezeichnet.

Produkte einen eher geringen Stellenwert einnehmen und stattdessen mit fortschrittlichen Simulationstechniken und hoher Integration geworben wird [306]. Im kommerziellen Bereich numerischer Strömungssimulation ist *Computational Steering* somit noch nicht sichtbar. Hingegen existieren im akademischen Umfeld zahlreiche Individuallösungen.

2.5.1 Monitoring mit einfacher Steuerungsmöglichkeit

Eine zentrale Rolle des *Computational Steerings* nimmt das Monitoring ein. Deswegen steht es im Vordergrund vieler aktueller Entwicklungen zur interaktiven Simulation von Fluiden. Ein häufiges Vorgehen besteht in der Kopplung der Strömungslöser mit modularen Visualisierungsumgebungen. Dabei ist eine Benutzerinteraktion mit der CFD-Simulation zur Laufzeit jedoch häufig nur eingeschränkt möglich.

Die gViz-Bibliothek wurde von Brodlie et al. [37] als Erweiterung der Visualisierungsumgebung IRIS Explorer entwickelt, um einzelne Komponenten in einer verteilten kollaborativen Umgebung auszuführen. Eine solche Komponente könnte beispielsweise eine Simulation sein, die zur Laufzeit Ergebnisdaten an die grafische Benutzeroberfläche übermittelt und darüber gesteuert werden kann. Brodlie et al. [33] setzten die gViz-Bibliothek unter anderem zur Untersuchung der Dispersion von Schadstoffen in der Luft bei unterschiedlichen Windverhältnissen ein. Die Einflussnahme auf die Simulation zur Laufzeit wird von der Bibliothek jedoch generell auf die Änderung einfacher Skalarwerte beschränkt.

Das RealityGrid-Projekt [39] hat sich zum Ziel gesetzt, das Konzept einer VR-basierten Forschungsumgebung unter Verwendung des Grids zu erweitern und auf diese Weise massive Rechenressourcen und Forschungseinrichtungen zu verbinden. Im Mittelpunkt der Entwicklung steht eine Bibliothek, mit deren Hilfe bestehende Anwendungen um *Computational Steering*-Fähigkeiten bei moderatem Aufwand erweitert werden können [222]. Dabei wird ein komponentenbasierter Ansatz mit einer expliziten Trennung von *Monitoring*- und Steuerungsfähigkeiten verfolgt, der es ermöglicht, beliebig leistungsstarke Hardware (von *High-End-Workstations* bis hin zu *Handheldcomputern*) als Benutzerschnittstelle zu integrieren. In [124] wird beispielsweise das *Monitoring* einer Simulation auf einem PDA (*Personal Digital Assistant*) gezeigt. Modifikationen von Simulationsparametern ermöglicht die Bibliothek auf zwei Weisen. Sie können, sofern ein gemeinsames Dateisystem verwendet wird, mittels Änderungen von Konfigurationsdateien erfolgen, oder nach vorheriger Registrierung der entsprechenden Parameter über SOAP-basierten (*Simple Object Access Protocol*) Nachrichtenaustausch. Zusätzlich dazu wird ein *Checkpointing*-Verfahren unterstützt, das es erlaubt, persistierte Zustände im Simulationsverlauf wiederherzustellen und zudem ein Migrieren einzelner Jobs zwischen verschiedenen Ressourcen ermöglicht. Das vielfach ausgezeichnete TeraGyroid-Projekt [24, 114] um die Gruppe von Coveney demonstriert die erfolgreiche Steuerung verschiedener CFD-Löser, darunter auch verschiedene LBM-basierte Verfahren, unter Einsatz der RealityGrid-Technologie [58]. Im Vordergrund steht dabei die Online-Visualisierung der Simulation. Die Benutzerinteraktion mit der Simulation erlaubt neben dem Starten, Stoppen und Pausieren der Berechnungen auch das Anpassen von physikalischen Parametern wie Relaxationszeit, Dichte und dem Schreibintervall zur Laufzeit. Ergebnisdaten werden zur Analyse an ein Erweiterungsmodul der AVS/Express-Visualisierungsumgebung gesendet. Als Alternative dazu wird im Rahmen des RealityGrid-Projektes eine *Monitoring*-Software basierend auf VTK (*The Visualization Toolkit*) [244] entwickelt.

2.5.2 Erweiterte Steuerungsmöglichkeiten

Kühner [154] präsentiert einen virtuellen Windkanal auf Basis der Lattice-Boltzmann-Methode, der eine Verbindung von Hochleistungsrechnern zur Simulation und VR-System auf Anwenderseite vorsieht. Die Anwendung erlaubt neben einer einfachen Anpassung von Skalarwerten auch die Manipulation der Geometie und Position von Hindernissen im Strömungsgebiet zur Laufzeit. Die Anwendung ist jedoch auf einfache Geometrien beschränkt und erfordert ein Vorberechnen der jeweiligen, zugrundeliegenden Berechnungsgitter.

Aufbauend auf dieser Arbeit stellt Wenisch [300] eine Komfortsimulation von Innenräumen vor, in der es möglich ist, beliebig komplexe Geometrien zur Laufzeit zu manipulieren und zu ergänzen, ohne dass eine Vorverarbeitung erforderlich ist. Die notwendige Anpassung des Berechnungsgitters verläuft dabei automatisiert ohne Zutun des Benutzers. Ein besonderer Aspekt der Arbeit besteht daher in der Optimierung der Algorithmen zur Gittergenerierung. Das Anwendungsframework wird von Borrmann [29, 30] um Mehrbenutzerfähigkeit erweitert, so dass kollaboratives Arbeiten ermöglicht wird. Geller demonstriert, dass das Verfahren grundsätzlich auch bei adaptiven, nicht-uniformen Gittern Anwendung finden kann [104].

Fahrig et al. [84, 150] beschreiben eine HVAC-Simulationsumgebung (*Heating, Ventilating and Air Conditioning*) zur Klimatisierung von Innenräumen. Dabei handelt es sich um einen gekoppelten Ansatz basierend auf einer LB-Simulation ausgeführt auf einem PC-Cluster, der Visualisierungsumgebung AVS/Express und der CAD-Anwendung AutoCAD [11]. Die Simulation unterstützt neben einer Interaktion des Anwenders auch eine automatisierte Regelung der Klimatisierung basierend auf Software-Agenten [296]. Diese Form der Steuerung wird von Vetter und Schwan [291] im Gegensatz zur interaktiven Steuerung durch den Benutzer (*Human-Interactive Steering*) als algorithmische Steuerung (*Algorithmic Steering*) bezeichnet.

COVISE (Collaborative Visualization Environment) gehört zu der Kategorie der modularen Visualisierungsumgebungen und wurde ursprünglich von Wierse et al. im Jahr 1993 vorgestellt [303]. Seitdem wird es in einer Serie von europäischen Forschungsprojekten entwickelt und findet unter anderem in den Bereichen Automobilbau, Luftfahrt und Maschinenbau Anwendung zur Gittergenerierung, Simulation und zum *Post-Processing*. Bei der Entwicklung wird insbesondere eine effiziente Netzwerkkommunikation und Integrationsfähigkeit von Umgebungen des Hochleistungsrechnens berücksichtigt [230, 302]. Der modulare Aufbau bietet zudem die Möglichkeit, weitere Funktionalität zu ergänzen. Mit Hilfe der Erweiterung COVER [229] unterstützt COVISE beispielsweise Methoden der virtuellen Realität (VR). Im Kontext einer immersiven VR-Umgebung wird COVISE unter anderem zur Fluid-Struktur-Interaktion (FSI) und numerischen Optimierung von Wasserturbinen eingesetzt [237]. Der Benutzer hat hier die Möglichkeit, zur Laufzeit der Simulation den Anstellwinkel der Turbine zu variieren und damit die Geschwindigkeit des Wasserstromes zu regulieren. COVISE unterstützt dabei eine möglichst intuitive Form der Interaktion durch die Entwicklung sogenannter *tangible interfaces* [16]. Die Simulation erfolgt dabei mit Hilfe des Strömungslösers FENFLOSS [299] basierend auf den Reynolds-gemittelten Navier-Stokes-Gleichungen.

2.6 Zusammenfassung und Fazit

Computational Steering stellt eine wertvolle Unterstützung im Simulationsprozess dar, mit dessen Hilfe Ingenieure und Wissenschaftler Berechnungen effizienter durchführen und Ergebnisse intuitiver analysieren können. Es trägt damit maßgeblich zum Verständnis physikalischer Zusammenhänge bei. Eine entscheidende Rolle in der Analyse nimmt dabei die wissenschaftliche Visualisierung ein.

Ein naheliegender Schritt in Richtung *Computational Steering* besteht daher in der Erweiterung modularer Visualisierungsumgebungen um *Monitoring*- und Steuerungsfähigkeiten. Hinsichtlich der Anwenderfreundlichkeit beinhaltet dieser Ansatz jedoch gewisse Einschränkungen. Die universellen Umgebungen sind im Allgemeinen nicht auf die speziellen Anforderungen des jeweiligen Anwendungsgebiets zugeschnitten. So bringen insbesondere die im CFD-Bereich anfallenden zeitabhängigen, großen Datenmengen die Systeme häufig an ihre Leistungsgrenzen. Zusätzlich erfordert der Umgang mit den komplexen modularen Systemen ein hohes Maß an Fachkompetenz. Wegen der teilweise begrenzten Erweiterungsfähigkeit der Systeme ist zudem die Interaktionsfähigkeit mit der Simulation häufig nur eingeschränkt oder gar nicht möglich. Im Allgemeinen wird es dabei nicht erreicht, die Möglichkeit zur intuitive Benutzerinteraktion mit allen Teilen des Simulationsprozesses in einer einzelnen Anwendungsumgebung zu vereinen. In der Regel sind dafür fachspezifische Individuallösungen erforderlich.

Insbesondere im Bereich der Strömungssimulation resultiert eine zusätzliche Herausforderung sowohl für Anwender als auch Entwickler aus dem Einsatz von traditionellen HPC-Systemen. Die Entwicklung von Standards im Bereich der *Middleware* Grid-Computings wie beispielsweise SAGA (*Simple API for Grid Applications*) kann zukünftig deutlich zur Reduzierung der Komplexität und Steigerung der Stabilität beitragen [315]. Generell reduzieren die hohen Anschaffungs- und Betriebskosten von traditionellen HPC-Systemen den potentiellen Anwenderkreis erheblich. In dieser Hinsicht kann jedoch Dank der Serviceorientierung des Grid-Computings in Zukunft eine breitere Zugänglichkeit erwartet werden. Als Mittel zur Kommunikation zwischen den Modulen hat die *Middleware* einen entscheidenden Einfluss auf die Qualität der Anwendung. Diese profitiert in erheblichem Maße von einer hohen Bandbreite und geringen Latenz. Während sich die Latenz insbesondere auf die Responsivität der interaktiven Anwendung auswirkt, beeinflusst die Bandbreite in erster Hinsicht die Qualität der visuellen Darstellung von Ergebnisdaten.

Viele der beschriebenen *Computational Steering*-Ansätze stellen zweifelsfrei sehr hochwertige Lösungen dar und tragen maßgeblich zum Verständnis komplexer ingenieurwissenschaftlicher Probleme bei. Dennoch kann zu Recht behauptet werden, dass heutige interaktive Umgebungen zur Strömungssimulation in der Mehrzahl weit davon entfernt sind, erstens weit verbreitet Anwendung zu finden und zweitens über eine einfache und intuitive Bedienbarkeit zu verfügen. Daher müssen Anwender heutiger Verfahren nicht nur Experten auf dem Gebiet der Strömungsmechanik sein, sondern auch über fundierte Kenntnisse im Bereich Hochleistungsrechnen, Visualisierung, Netzwerkkommunikation und Softwareentwicklung verfügen.

3 VIRTUALFLUIDS INTERACTIVE - Übersicht und Anforderungen

Im Mittelpunkt dieser Arbeit steht die Entwicklung der interaktiven Simulationsumgebung zur Berechnung von Strömungen VIRTUALFLUIDS INTERACTIVE. Zur Einordnung und Förderung des Verständnisses der nachfolgenden Kapitel wird die Anwendung im Rahmen dieses Kapitels vorgestellt. Dazu erfolgt zunächst eine kurze Erläuterung von Funktion und Aufbau, die gleichzeitig die wichtigsten Aspekte dieser Arbeit hervorhebt. Eine detaillierte Betrachtung der einzelnen Teilaspekte schließt sich in den nachfolgenden Kapiteln an.

3.1 Anwendungsfunktionalität

Allgemein besteht die Aufgabe einer interaktiven Simulationsumgebung darin, die Komplexität des Simulationsverfahrens vor dem Anwender zu abstrahieren. Als Abstraktionsschicht zwischen dem Benutzer und der Simulation übernimmt die interaktive Umgebung die Aufbereitung und Kommunikation von Daten in eine dem Menschen verständliche Darstellungen und, vice versa, Benutzerinteraktionen in Eingangsdaten für die Simulation. Diese Zusammenhänge werden in Abbildung 3.1 veranschaulicht. Der dabei auftretende Datenfluss des Steuerungsprozesses innerhalb der Anwendung ist analog in Abbildung 3.2 dargestellt. Die Abbildungen verdeutlichen zudem

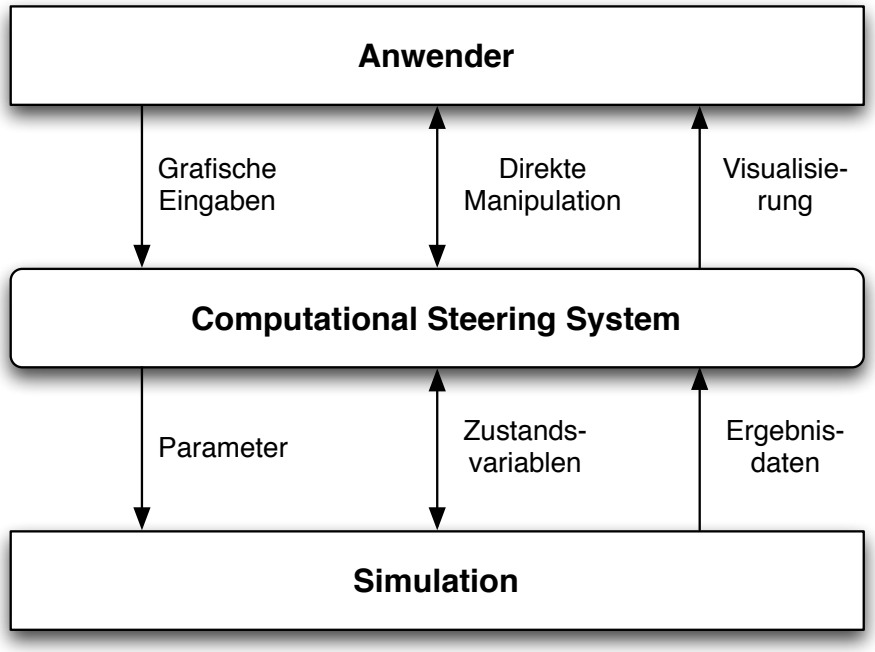


Abbildung 3.1: *Die interaktive Simulationsumgebung als Abstraktionsschicht zwischen Anwender und Simulation.*

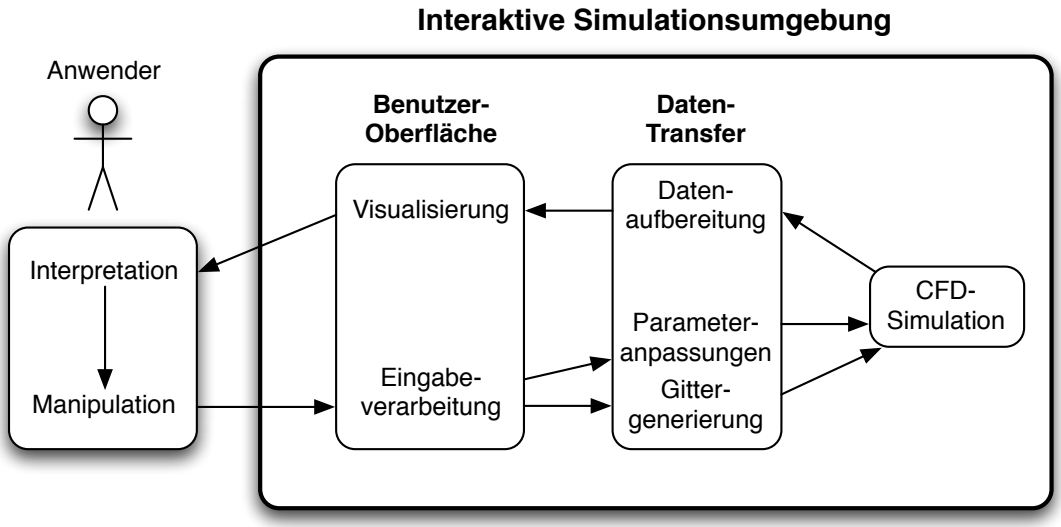


Abbildung 3.2: *Veranschaulichung des Datenflusses innerhalb von* VIRTUALFLUIDS INTERACTIVE *basierend auf [195].*

den wesentlichen Aufbau der Anwendung, bestehend aus drei Komponenten: Benutzerschnittstelle, Simulation und Datentransfer beziehungsweise Datenaufbereitung. Diese Komponenten sind in VIRTUALFLUIDS INTERACTIVE anders als bei anderen Lösungen in eine einzelne Rahmenanwendung integriert. Dabei ist insbesondere die Integration von Benutzerschnittstelle und Simulationskomponente hervorzuheben, ohne die es laut Johnson [133] und Parker [216] letztlich nicht möglich ist, das gesamte Potential des *Computational Steering*-Paradigmas zu realisieren.

3.1.1 Intuitives interaktives Bedienkonzept

Eine besondere Herausforderung bei der Entwicklung einer *Computational Steering*-Umgebung stellen Interaktivität und Bedienkonzept dar. Die Möglichkeit zur direkten Interaktion mit sowohl der Visualisierung als auch der laufenden Simulation wirkt sich dabei im Allgemeinen deutlich unterstützend auf den explorativen Simulationssprozess aus. Die Interaktionsmöglichkeiten sollten indes so intuitiv und natürlich wie möglich gestaltet werden [204, 254]. Eine besondere Schwierigkeit ergibt sich, wenn die Benutzeroberfläche neben konventionellen 2D-Bedienelementen auch Elemente zur 3D-Darstellung und Interaktion enthält. Um das Verständnis zu fördern, erfolgt die Darstellung der Simulationsergebnisse in VIRTUALFLUIDS INTERACTIVE kombiniert mit der Geometrie des Strömungsgebiets in einem zentralen Anwendungsfenster. Das zentrale Element dient zum einen der effektiven und effizienten interaktiven Darstellung von Ergebnisdaten, erlaubt aber auch eine direkte Manipulation der Strömungsgebietsgeometrie. Dieses Interaktionsparadigma ersetzt die Eingabe von Befehlen über die Tastatur durch eine direkte Manipulation des dargestellten Objektes, beispielsweise mit Hilfe einer Maus [252]. Obwohl eine intuitive direkte Manipulation viele Vorteile bietet [254], ist sie nicht generell für jede Form von Parametern sinnvoll und zudem nur mit eingeschränkter Genauigkeit möglich. Daher kann in VIRTUALFLUIDS INTERACTIVE alternativ auch eine indirekte, indes präzise, Manipulation von Parametern über klassische 2D-Eingabeelemente erfolgen.

3.1.2 Interaktive geometrische Modellierung

VIRTUALFLUIDS INTERACTIVE kann als ein virtueller Strömungskanal angesehen werden, dessen Fluss wahlweise durch Vorgabe eines Druckgradienten oder eines variablen Geschwindkeitsprofils an den Rändern des Kanals induziert wird. Innerhalb des Strömungsgebiets können neben primitiven Geometrien wie Kugeln und Quadern auch komplexe facettierte Dreiecksgitter als Hindernisse platziert werden. Mit Hilfe triangulierter Geometrien, die beispielsweise im STL-Format (*Surface Tesselation Language*) vorliegen, lässt sich die Oberfläche beliebiger Formen quadratisch genau annähern und somit in VIRTUALFLUIDS INTERACTIVE importieren. Jede dieser Geometrien kann vom Benutzer innerhalb der Anwendung intuitiv manipuliert werden. Dabei werden Transformationen wie Translation, Rotation, Skalierung und Scherung unterstützt. Die interaktive geometrische Modellierung des Strömungsgebiets verläuft häufig iterativ. Daher ist es möglich, jede angewandte Transformation im Fall eines unerwünschten Ergebnisses zu widerrufen. Das kontinuierliche geometrische Modell wird anschließend in eine diskrete Form überführt und mit Randbedingungen versehen, um als Basis für eine Simulation zu dienen. Dabei kommen performante Algorithmen zum Einsatz, die auf den Arbeiten von Bindick [23] und Freudiger [94] basieren. Das Ergebnis ist in diesem Fall ein uniformes, kartesisches Berechnungsgitter. Die Gittergenerierung verläuft vollständig automatisiert, so dass es möglich ist, Änderungen an der Geometrie des Strömungsgebiets auch zur Laufzeit vorzunehmen. Auf Grund leistungsfähiger Algorithmen kommt es dabei je nach Gebietsgröße lediglich zu geringen Verzögerungen der fortschreitenden Simulation. Neben Geometrie und Randbedingungen bestimmen Materialparameter des Fluides wie Dichte und Viskosität das Strömungsverhalten. Zur Unterstützung des Anwenders bietet VIRTUALFLUIDS INTERACTIVE eine Vorauswahl an Materialien wie beispielsweise Luft und Wasser bei verschiedenen Temperaturen. Generell können die Materialparameter jedoch beliebig gewählt und variiert werden.

3.1.3 Numerische Strömungssimulation

Als Verfahren zur Simulation von Strömungen wird in VIRTUALFLUIDS INTERACTIVE die Lattice-Boltzmann-Methode (LBM) eingesetzt [149, 267] (vgl. Abschnitt 5.1). Dabei handelt es sich um ein explizites Finite-Differenzen-Schema zur näherungsweisen Lösung der Navier-Stokes-Gleichungen im Limes inkompressibler Strömungen. Die Methode arbeitet im Allgemeinen auf kartesischen Gittern, an deren Knoten in jedem Zeitschritt ein sogenannter lokaler Kollisionsschritt und ein darauf folgender Propagationsschritt durchgeführt werden. Dabei wird an jedem Knoten die lokale Verteilung virtueller mikroskopischer Teilchen anhand statistischer Wahrscheinlichkeitsfunktionen bestimmt (*Kollision*) und diese in einem anschließenden Schritt zu den Nachbarknoten propagiert (*Propagation*). Auf Grund der hohen Lokalität des Verfahrens kann der Algorithmus sehr effizient parallelisiert werden. Im Allgemeinen erlaubt die Lattice-Boltzmann-Methode zudem eine nahezu stufenlose Skalierung der Gitterauflösung, wodurch dem Anwender eine feine Balancierung von erzielbarer Genauigkeit und Ressourcenbedarf ermöglicht wird. Generell unterstützt VIRTUALFLUIDS INTERACTIVE die Berechnung stationärer, laminarer und auch turbulenter Strömungen. Zur Berechnung turbulenter Strömungen kann optional ein LES-Turbulenzmodell (*Large Eddy Simulation*) aktiviert werden [151, 224]. Die Stabilität und Genauigkeit des Verfahrens kann zusätzlich durch die Wahl des Differenzensterns und des Kollisionsmodells beeinflusst werden.

3.1.4 Interaktive Exploration

VIRTUALFLUIDS INTERACTIVE bietet verschiedene Strategien für die interaktive visuelle Aufbereitung komplexer Strömungsdaten an. Dazu zählen sowohl globale als auch lokale Visualisierungsmethoden, mit deren Hilfe die zeitabhängigen makroskopischen Größen des Strömungsfeldes qualitativ und quantitativ analysiert werden können. Möglich ist sowohl eine Exploration der skalaren Dichte (des Drucks) und der Wirbelstärke als auch des vektoriellen Geschwindigkeitsfelds. Die interaktive Visualisierung stellt damit ein zentrales Hilfsmittel für die Analyse und Präsentation von Ergebnissen laufender Simulationen dar, die in vielerlei Hinsicht unterstützend wirkt. Kommunikation, Diskussion, Fehlersuche, Verständnis und Intuition sind nur einige Aspekte des Simulationsprozesses, die positiv beeinflusst werden. Auf Grund der iterativen Eigenschaft des Visualisierungsprozesses kann die Art der Visualisierung jederzeit auf intuitive Art angepasst werden. Zudem ist es möglich, die Ausführung der Simulation bei Bedarf zu pausieren und sowohl die gerenderte Darstellung als auch die Simulationsergebnisse für eine spätere Präsentation und Analyse im *Post-Processing* (in variabler Auflösung) zu speichern. Das Persistieren der Ergebnisdaten basiert dabei auf dem HDF5-Datenmodell (*Hierarchical Data Format*) [275] und erlaubt einen flexiblen Datenzugriff. VIRTUALFLUIDS INTERACTIVE unterstützt neben dem *Post-Processing* von uniformen Ergebnisdaten auch die Analyse von Daten basierend auf hierarchischen Gittern. Zu diesem Zweck stellt die Anwendung Schnittstellen für den Datenaustausch auf Basis des HDF5-Formats bereit (vgl. Abschnitt 6.5.3). Dabei kommen verschiedene Reduktionsverfahren in Zeit und Raum zum Einsatz, um die häufig erheblichen Datenmengen zu reduzieren.

3.1.5 GPU-Computing als Alternative zu traditionellem HPC

Generell ist eine Benutzerinteraktion mit laufenden Simulationen nur dann zweckmäßig, wenn die Berechnungen auch bei relevanten Gitterauflösungen hinreichend schnell evolvieren. Wegen der hohen Leistungsanforderungen des numerischen Verfahrens konnte dieser Forderung bislang nur durch den Einsatz traditioneller Hochleistungsrechner nachgekommen werden. Dabei ist in der Regel ein Kompromiss aus Genauigkeit des Simulationsverfahrens und verschiedenen Leistungsfaktoren der Hardwareinfrastruktur einzugehen. Limitierend auf die Interaktivität der *Computational-Steering*-Umgebung wirkt sich hier im Allgemeinen die Bandbreite und Latenz notwendiger Netzwerkverbindungen aus. Ein Trend der heutigen Zeit besteht jedoch darin, die immense parallele Rechenleistung von Grafikprozessoren (GPUs) für universelle Rechenaufgaben zu verwenden. Obwohl das sogenannte GPU-Computing auch schon in der Vergangenheit Anwendung gefunden hat, wurde das Verfahren erst mit steigender Programmierbarkeit der GPUs und vor allem durch die Veröffentlichung der *Compute Unified Device Architecture* (CUDA) [206], einer deutlich vereinfachten Programmierschnittstelle, insbesondere im Umfeld des wissenschaftlichen Rechnens populär. Tölke und andere haben erfolgreich gezeigt, dass das GPU-Computing auch auf die Lattice-Boltzmann-Methode angewendet werden kann [279, 281]. VIRTUALFLUIDS INTERACTIVE setzt auf dieser Technologie auf und erlaubt es, die Simulationen auf einer oder auch parallel auf mehreren CUDA-fähigen Grafikkarten interaktiv durchzuführen. Anders als im traditionellen Hochleistungsrechnen üblich, ist es auf diese Weise möglich, die Anwendung auf einem einzelnen Arbeitsplatzrechner auszuführen und dabei trotzdem eine Rechenleistung zu erzielen, die ansonsten nur von Hochleistungsrechnern erreicht werden kann.

3.1.6 Abgrenzung zu bestehenden Systemen

Durch den Einsatz von Grafikkarten für rechenintensive Simulationen ergeben sich im Vergleich zu traditionellen *Computational Steering*-Umgebungen sowohl aus Entwickler- als auch aus Anwendersicht deutliche Vorzüge. So trägt die GPU-Technologie dazu bei, die anfallenden Kosten für Hardware und Betrieb sowie die Komplexität der Anwendung deutlich zu reduzieren und gleichzeitig die Benutzerfreundlichkeit zu steigern. Heute verfügbare, speziell für HPC-Anwendungen ausgewiesene Grafikkarten können hinsichtlich ihrer nominellen Rechenleistung mit kleinen PC-Clustern konkurrieren. Dennoch sind sie vergleichsweise günstig in der Anschaffung und weisen zudem einen deutlich geringeren Energieverbrauch auf. War das Hochleistungsrechnen bislang in der Regel finanzstarken Unternehmen und Bildungseinrichtungen vorbehalten, so erschließen sich GPU-basierte Systeme dank des Kostenvorteils allgemein einer signifikant größeren Zielgruppe. Zusätzlich profitieren sowohl Anwender als auch Entwickler gleichermaßen von der Tatsache, dass GPU-basierte Anwendungen im Gegensatz zu traditionellen HPC-Ansätzen nicht notwendigerweise in einer verteilten Umgebung ausgeführt werden müssen. Vielmehr ist ein entsprechend ausgerüsteter Arbeitsplatzrechner in bestimmten Fällen ausreichend leistungsfähig, um hinreichend genaue Lösungen in angemessener Zeit berechnen zu können. In diesem Fall reduziert sich die Komplexität der Anwendung erheblich, da eine Interprozesskommunikation entfällt und somit auf zusätzliche Middleware für die Kommunikation verzichtet werden kann[1]. Für den Endanwender führt die Reduktion auf eine einzelne *Workstation* zu einem erheblichen Komfortgewinn. Die hohe Lokalität der Anwendungsumgebung hat zudem einen erheblichen Einfluss auf die Anwendungsqualität. Beispielsweise ist es nicht erforderlich, Ergebnisdaten durch ein Netzwerk zu kommunizieren, dessen Latenz und Bandbreite in der Regel einen Flaschenhals darstellen [164, 300, 301, 315]. In Folge dessen profitieren *Computational Steering*-Systeme von einer deutlichen Steigerung der Responsivität und Aktualisierungsrate (vgl. Abschnitt 3.2.1). Klassische Ansätze basieren zumeist auf heterogenen Umgebungen, die dem Benutzer Kompetenz im Umgang mit diversen Systemen, insbesondere aus dem HPC-Bereich, abverlangen. Im Gegensatz dazu stellt VIRTUALFLUIDS INTERACTIVE eine hoch integrierte Simulationsumgebung bereit, deren Benutzeroberfläche zu großen Teilen an herkömmliche Anwendungen angelehnt ist und ein einheitliches sowie weitestgehend intuitives Bedienkonzept verfolgt.

VIRTUALFLUIDS INTERACTIVE grenzt sich damit insbesondere in Hinblick auf die Benutzerfreundlichkeit und anfallenden Kosten von bestehenden Systemen ab. Dabei eignet sich die Anwendung weniger für großskalige Produktionsläufe als vielmehr für ein *Rapid Prototyping* und eine hoch interaktive Fehlersuche (*Debugging*). VIRTUALFLUIDS INTERACTIVE zeigt damit ein Potential auf, das basierend auf aktuellen GPU-Technologien allgemein zu einer weiteren Verbreitung des *Computational Steering*-Paradigmas und einem einhergehenden Effizienzgewinn beitragen kann.

3.2 Anforderungen an eine interaktive Simulationsumgebung

Im Allgemeinen wird eine interaktive Simulationsumgebung nur dann erfolgreich sein, wenn sie für Ingenieure und Wissenschaftler nachweislich ein hilfreiches Werkzeug darstellt [133, 216]. Als solches übernimmt eine *Computational Steering*-Anwendung im Bereich der numerischen Strömungs-

[1] Dieser Vorteil relativiert sich in gewissem Maße, da durch den Einsatz der CUDA-Technologie zusätzliche Komplexität eingeführt wird.

simulation vor allem die Aufgabe, eine informative Darstellung der Ergebnisdaten zu gewährleisten sowie eine intuitive Benutzerinteraktion mit der Simulation und der Visualisierung zu ermöglichen. Als Mittel zur Kommunikation zwischen Benutzer und Simulation sollte die Anwendungsumgebung den Anwender dabei bestmöglich involvieren, um ein Höchstmaß an Aufmerksamkeit zu erwirken. Eine Schlüsselrolle nimmt dabei die Interaktionsmöglichkeit in Echtzeit ein [45]. Es zeigt sich, dass die Umsetzung dieser Aufgaben eine besondere Herausforderung darstellt, da sich vor allem sehr hohe Anforderungen an die Leistungsfähigkeit des Systems ergeben und ein 3D-Interaktionsparadigma erforderlich ist [44]. Dennoch gibt es kein allgemein gültiges Vorgehen, um insbesondere den Leistungsanforderungen garantiert nachzukommen [44]. Stattdessen ist im Allgemeinen ein Kompromiss aus Geschwindigkeit und Genauigkeit erforderlich. Dabei ist es von besonderer Bedeutung, dass dieser Kompromiss vom Benutzer getroffen werden kann. Die mitunter konkurrierenden Eigenschaften eines erfolgreichen *Computational Steering*-Systems werden im Folgenden näher betrachtet.

3.2.1 Grafische Darstellung und intuitive Interaktion

In einer interaktiven Umgebung nimmt das Visualisierungssystem einen besonderen Stellenwert ein. Dieses zeichnet sich einerseits durch seine Darstellungsqualitäten sowie andererseits durch seine Interaktionsfähigkeiten aus.

Darstellungsqualität

Die allgemein übliche Methode zur Analyse der Ergebnisdaten ist die wissenschaftliche Visualisierung. Dabei werden allgemein formuliert Methoden der Computergrafik verwendet, um komplexe numerische Daten auf grafische Repräsentationen abzubilden. Im Gegensatz zur Computergrafik, die versucht, Objekte der realen Welt realistisch abzubilden, orientiert sich die Darstellung der wissenschaftlichen Visualisierung eher an einer möglichst zweckmäßigen Repräsentation abstrakter Daten. Die objektive Bewertung einer Visualisierung hinsichtlich ihrer Qualität ist jedoch diffizil. Ein Ansatz besteht in der Beurteilung anhand ihrer Expressivität, Effektivität und Angemessenheit (vgl. Abschnitt 6.1.5) [247, 306].

Aktualisierungsrate

Die Simulation transienter Strömungsphänomene ist ein inkrementeller Prozess, bei dem kontinuierlich Zwischenergebnisse erzeugt werden. Diese werden für das Monitoring einer interaktiven Umgebung in definierten Zeitschrittintervallen visualisiert. Mit Aktualisierungszeit (*Update Time*) wird in dieser Arbeit die Zeitspanne von Beginn eines Zeitschrittintervalls bis zur Darstellung auf dem Bildschirm bezeichnet. Je geringer diese Zeitspanne ist, desto höher ist die Aktualisierungsrate (*Update Rate*). Neben dem Laufzeitverhalten der Simulation haben die Kommunikation der Ergebnisdaten und die unter Umständen sehr rechenintensive Visualisierung einen entscheidenden Einfluss auf die Aktualisierungsrate. Diese ist insbesondere bei zeitkritischen Anwendungen wie interaktiven Simulationen von erheblicher Bedeutung. Im Fall zu geringer Leistungsfähigkeit büßt eine *Computational Steering*-Umgebung einen Großteil ihrer Vorteile ein [165].

Intuitive Interaktion

Die wichtigste Voraussetzung für eine effektive Exploration der Simulationsmodelle besteht darin, dem Anwender effiziente Interaktionsmethoden zur Verfügung zu stellen, die es ihm ermöglichen, sowohl Parameter der Simulation als auch der Visualisierung anzupassen. Letzteres trägt insbesondere bei komplexen, zeitabhängigen Datensätzen entscheidend dazu bei, qualitative und quantitative Informationen zu entnehmen (Zitat [44]: »Real-time interaction encourages exploration.«). Anders als bei herkömmlichen grafischen Benutzeroberflächen existiert jedoch keine Standardisierung für die Interaktion mit interaktiven 3D-Visualisierungssystemen [17]. Umso erforderlicher ist es, die Interaktionsmechanismen intuitiv zu gestalten, so dass diese im Idealfall keiner weiteren Erklärung bedürfen [204, 254]. Die meisten heute frei verfügbaren und auch kommerziellen CFD-Software-Pakete entsprechen jedoch nicht dieser Anforderung [170].

Bildrate

Während einer Benutzerinteraktion mit dem Visualisierungssystem ist es erforderlich, dass die Darstellung in Echtzeit animiert wird. Dabei ist eine Mindestbildrate von 14-16 Bildern pro Sekunde notwendig. Aktuelle Grafikkarten erreichen diesen Wert in der Regel auch bei komplexen Darstellungen von hoher Qualität. Dennoch ist es ratsam, den Detailreichtum der Darstellung gering zu halten, um einerseits die Bildrate auf ein Maximum zu steigern und andererseits ein hinderliches Überladen der Szene mit Informationen zu vermeiden [17].

Responsivität

Die Responsivität bezieht sich auf das Zeitintervall zwischen einer Benutzerinteraktion und der Antwort des Systems in Form einer Bildschirmausgabe. Die Aufrechterhaltung der Responsivität einer interaktiven Simulationsumgebung stellt eine besondere Herausforderung dar [45]. Eine gewisse Latenz wird sich niemals vermeiden lassen. Es zeigt sich jedoch, dass Benutzer bei Zeitintervallen von mehreren Sekunden dazu neigen anzunehmen, das System habe ihren Befehl nicht registriert. In diesem Fall tendieren die Anwender in der Regel dazu, das Kommando entweder zu wiederholen oder anzunehmen, das System sei defekt [204].

3.2.2 Allgemeine Anforderungen

Im Allgemeinen ist eine hohe Ausführungsgeschwindigkeit bei jeder Form von Anwendung wünschenswert. Im Hinblick auf die vorausgehend betrachteten Kriterien einer *Computational Steering*-Anwendung gilt es jedoch zu beachten, dass die Leistungsfähigkeit des Systems einen unmittelbaren Einfluss auf den Erfolg des Explorationsprozesses besitzt. Sie stellt damit, ebenso wie eine hohe Benutzerfreundlichkeit und die Möglichkeit große Datenmengen zu verarbeiten, eine essentielle Anforderung an eine interaktive Simulationsumgebung dar [43, 44]. Neben den bislang betrachteten

nicht-funktionalen Anforderungen[2] sind allgemein auch die funktionalen Anforderungen[3] für die Entwicklung relevant. Auf eine umfassende Beschreibung der funktionalen Anforderungen soll an dieser Stelle jedoch verzichtet werden. Stattdessen sei darauf hingewiesen, dass individuelle Anwender unterschiedliche Anforderungen an die Funktionalität einer *Computational Steering*-Umgebung haben. Aus diesem Grund sollte das System Schnittstellen für eigene Erweiterungen bereitstellen. Auf diese Weise kann die Zweckdienlichkeit des Systems sichergestellt werden, ohne das Basissystem mit einem großen, unübersichtlichen Funktionsumfang zu überladen. Johnson und Parker weisen zudem darauf hin, dass die Erweiterbarkeit des Systems einen entscheidenden Einfluss auf den Erfolg einer Anwendung im Bereich des *Computational Steerings* besitzt [133, 216]. Bis dato haben sich Forschungsarbeiten auf diesem Gebiet im Wesentlichen jedoch auf die Interaktionsfähigkeiten und die Implementierung genereller Machbarkeitsstudien fokussiert. Themen wie Anwendungsqualität und Benutzerfreundlichkeit wurde bislang nur wenig Aufmerksamkeit geschenkt [133, 216]. Da diese Aspekte insbesondere für den Erfolg des Paradigmas entscheidend sind, werden sie in der vorliegenden Arbeit in den Fokus gerückt.

[2] Nicht-funktionale Anforderungen sind nach Sommerville [258] Anforderungen, welche die durch das System zu leistenden Funktionen nicht direkt betreffen.

[3] Die funktionalen Anforderungen an ein System beschreiben nach Sommerville [258] die Funktionalität oder die Dienste, die von einem System erwartet werden.

4 Basisarchitektur

Im Rahmen des folgenden Kapitels wird die Basisarchitektur von VIRTUALFLUIDS INTERACTIVE vorgestellt. Dazu erfolgt zunächst eine Betrachtung der Aspekte eines qualitativ hochwertigen Softwareentwurfs, da dieser maßgeblich zur allgemeinen Anwendungsqualität beiträgt.

4.1 Anwendungsqualität

> »Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction and skillful execution; it represents the wise choice of many alternatives.«
> William A. Foster

Zu den in Abschnitt 3.2 genannten Anforderungen zählt unter anderem die Anwendungsqualität. Die Merkmale zur Beurteilung dieser Qualität werden nach McCall [186] in zwei Kategorien eingeteilt. Dabei wird hinsichtlich sogenannter externer und interner Qualitätsfaktoren unterschieden. Während die externe Qualität darüber Aufschluss gibt, inwieweit die Anwendung den Anforderungen und Bedürfnissen des Anwenders entspricht, sind für Entwickler insbesondere die internen Qualitätsmerkmale von Interesse. Bei externen wie internen Qualitätsmerkmalen kommt es gleichermaßen zu Überschneidungen, die sich allerdings hinsichtlich einiger Bedeutungsnuancen unterscheiden. Es existiert jedoch eine implizite Abhängigkeit einzelner Faktoren, wodurch der Versuch, bestimmte Merkmale zu maximieren, unweigerlich zu Konflikten führt. Die Balancierung dieser Aspekte zeichnet die Softwareentwicklung als eine Ingenieurdisziplin aus [187].

4.1.1 Externe Anwendungsqualität

Aus Sicht des Benutzers sollte sich eine Anwendung nach gängiger Meinung insbesondere hinsichtlich folgender Kriterien, den sogenannten externen Qualitätsfaktoren, auszeichnen [27, 186, 187, 191]:

- **Korrektheit:** Eigenschaft einer Anwendung, sich konform zu der Spezifikation zu verhalten, die sich aus den formalen Anforderungen ableitet.
- **Robustheit:** Eigenschaft einer Anwendung, auch außerhalb der Spezifikation stabil zu bleiben und Fehlbedienungen des Anwenders zu tolerieren.
- **Kompatibilität:** Maß an Flexibilität, mit der sich eine Anwendung ohne Änderungen an andere Anwendungen oder Umgebungen anpassen kann.
- **Effizienz:** Minimale Anforderungen an Hardware-Ressourcen wie CPU-Zeit und Arbeitsspeicher.

- **Benutzerfreundlichkeit:** Maß für die Leichtigkeit, mit der Benutzer ein System erlernen und verwenden können.
- **Zuverlässigkeit:** Fähigkeit des Systems, seine Funktionen mit einer geringen Ausfallrate unter definierten Bedingungen auszuführen.
- **Genauigkeit:** Grad der mathematischen Genauigkeit speziell bei quantitativen Aussagen.
- **Integrität:** Grad an Datensicherheit und Schutz vor unauthorisierten Zugriffen.

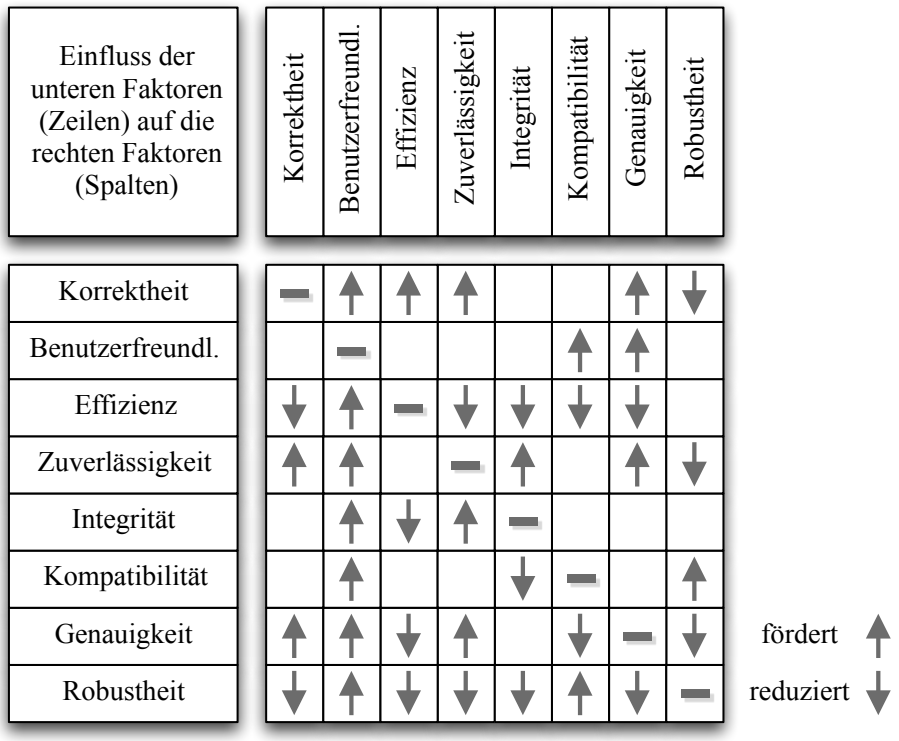


Abbildung 4.1: *Abhängigkeit externer Qualitätsfaktoren. Hinsichtlich der Softwarequalität kann die Betonung eines Merkmals andere Charakteristiken einer Computational Steering-Umgebung beeinflussen [187].*

4.1.2 Interne Anwendungsqualität

Der Schlüssel zum Erzielen einer hohen externen Qualität liegt in den internen Qualitätsfaktoren [191]. Sie umfassen Aspekte der Software, die in erster Linie Belange der Entwicklung betreffen. In der Literatur werden insbesondere folgende Merkmale beschrieben [27, 186, 187, 191]:

- **Wartungsfreundlichkeit:** Möglichkeit, Funktionalität der Anwendung anzupassen, um z.B. Fehler zu korrigieren und die Performance zu optimieren.
- **Flexibilität:** Anpassungsmöglichkeit des Systems an Anforderungen, für die es ursprünglich nicht entworfen wurde.
- **Wiederverwendbarkeit:** Möglichkeit, Teile der Anwendung wieder zu verwenden.

- **Portierbarkeit:** Integrationsmöglichkeit in veränderte Umgebungen.
- **Lesbarkeit:** Lesbarkeit des Quelltextes insbesondere auf Detailebene.
- **Verständlichkeit:** Kohärente Verständlichkeit des Systems sowohl auf hoher Abstraktionsebene als auch auf Detailebene.
- **Testbarkeit:** Möglichkeit zur Überprüfung, dass das System die spezifischen funktionalen Anforderungen erfüllt.

4.1.3 Komplexität vs. Qualität

Den Qualitätsfaktoren entgegen wirkt die Komplexität, die, wie Dijkstra [75] anmerkt, bei Softwaresystemen inhärent ist. Laut Booch [28] hat diese Komplexität vier Ursachen: Die Komplexität der Problemdomäne, die Schwierigkeit, den Entwicklungsprozess handzuhaben, die mögliche Flexibilität von Software und die Charakterisierung des Verhaltens diskreter Systeme. Aus immer umfangreicheren realen Problemstellungen folgt damit ein kontinuierliches Komplexitätswachstum der Softwaresysteme. Je komplexer jedoch das System ist, desto eher droht ein Versagen [221]. Die Aufgabe eines hochwertigen Softwareentwurfs besteht daher in erster Linie in der Beherrschung eben jener Komplexität [187, 233].

Die Objektorientierung (OO) ist ein erprobtes Paradigma zur Organisation der Komplexität von Software [28, 49, 93, 157, 180, 187, 191]. Die Methoden der objektorientierten Analyse (OOA) und des Entwurfs (OOD) (*Object-oriented Analysis and Design*) bilden dabei einen ingenieurmäßigen Lösungsansatz, der von den Anforderungen über den Entwurf zu einer Implementierung führt. Das Ergebnis des Entwurfs sind Modelle, die es ermöglichen, Strukturen abzuleiten, konkurrierende Anforderungen abzuwägen und eine Grundlage für die Implementierung darstellen. Das Ableiten von Modellen ist ein weit verbreiteter Ansatz innerhalb der Ingenieurdisziplinen. Das allgemeine Vorgehen basiert dabei im Wesentlichen auf den Prinzipien der Dekomposition, Abstraktion und Hierarchie [266]. Jedes Modell innerhalb des Entwurfs repräsentiert einen bestimmten Aspekt des abzubildenden Systems, wobei im Allgemeinen angestrebt wird, neue Modelle auf bereits erprobten existierenden Lösungen aufzusetzen. Modelle ermöglichen es den Ingenieuren, ihre Systeme unter kontrollierten Bedingungen hinsichtlich zu erwartender und unerwarteter Situationen zu erproben und bei Bedarf zu korrigieren. Die grundlegenden Konzepte der objektorientierten Modellbildung und Organisation von Softwarekomplexität werden basierend auf den Überlegungen von Booch [28] im Folgenden charakterisiert. Für eine detaillierte Betrachtung der Methodiken objektorientierter Softwareentwicklung wird auf einschlägige Referenzen wie beispielsweise [28, 49, 157, 191] verwiesen.

4.1.4 Basiskonzepte organisierter Komplexität

Um die Komplexität von Software mit Hilfe des objektorientierten Entwurfsprozesses zu organisieren, bedarf es der Einbeziehung einer Vielzahl an Aspekten. Dijkstra [75] weist jedoch darauf hin, dass es dem menschlichen Geist unmöglich ist, die inhärente Komplexität moderner Software als Ganzes aufzufassen. Experimente von Psychologen wie Miller [192] zeigen, dass der Mensch lediglich dazu imstande ist, in etwa sieben Informationen simultan zu erfassen.

Dekomposition

Eine seit langem bekannte Technik zur Bewältigung der Komplexität ist daher “divide et impera” (teile und herrsche) [76], bei der ein komplexes System so lange in kleinere Teile untergliedert wird, bis es der Kapazität der menschlichen Kognition genügt. Wie Parnas [220] beschreibt, adressiert eine geschickte Dekomposition direkt die inhärente Komplexität von Software durch eine Unterteilung des Zustandsraumes. Auf diese Weise wird es möglich, jede Ebene des Systems reduziert auf wenige Elemente verständlich zu machen. Dieses strategische Vorgehen wird auch als *Top-Down*-Ansatz bezeichnet und auf elegante Weise in der Arbeit von Wirth [307] erläutert.

Abstraktion

Wie Wulf [316] beschreibt, besitzt der menschliche Verstand die mächtige Fähigkeit, der Komplexität durch Abstraktion zu begegnen. Zwar ist die mögliche Anzahl der gleichzeitig aufzufassenden Informationen dabei nach wie vor begrenzt, dank Abstraktion wird es jedoch möglich, den semantischen Gehalt der Informationseinheiten deutlich zu erhöhen.

Hierarchie

Eine zusätzliche Erhöhung des semantischen Gehalts einzelner Informationseinheiten wird durch Einführung von Hierarchien möglich. Sie sind für die Entwurfsphase insbesondere wegen ihrer Eigenschaft, Verhaltens- und Kommunikationsstrukturen zwischen den Informationseinheiten zu verdeutlichen, von großer Bedeutung. Die einzelnen Hierarchieebenen repräsentieren dabei im Allgemeinen unterschiedliche Abstraktionsniveaus, die in der Regel aufeinander aufbauen, jedoch unabhängig voneinander verständlich sind. Die einzelnen Elemente einer Hierarchieebene interagieren dabei auf klar definierte Weise.

4.1.5 Schlüsselelemente des objektorientierten Entwurfs

Ausgehend von den allgemeinen Konzepten zur Organisation der Komplexität leiten sich die folgenden Schlüsselelemente objektorientierter Modelle ab:

Modularisierung

Die Anwendung der Dekomposition im objektorientierten Entwurf führt zu einer Modularisierung des Softwaresystems in einzelne Informationseinheiten[1]. Eine Dekomposition in vollständig unabhängige Module ist dabei in der Regel jedoch nicht zu erzielen. Stattdessen existiert eine Bindung zwischen den Komponenten, die jedoch, wie Simon [255] beschreibt, schwächer ausgeprägt ist als die Bindungen innerhalb der Module. Eine Folge der Modularisierung ist somit die Trennung der höherfrequenten Dynamiken interner Modulstrukturen von niederfrequenten Dynamiken der Kommunikation zwischen den Komponenten. Diese Separation der äußeren und inneren Interaktionen

[1] Informationseinheiten können z.B. durch Routinen, Klassen oder Pakete abgebildet werden [187]. In einem rein objektorientierten Ansatz jedoch werden Module ausschließlich von Klassen repräsentiert [191].

bewirkt eine klare Trennung der Anliegen (SoC) (*Separation of Concerns* [77]) innerhalb des Systems und ermöglicht damit eine annähernd isolierte Betrachtung der einzelnen Bestandteile. Mit Hilfe der Modularisierung wird somit das Erstellen komplexer Softwaresysteme durch Verbindung autonomer Komponenten in einfachen kohärenten Strukturen erleichtert [191].

Kapselung

Eine explizite Abgrenzung einzelner Module wird unter anderem durch das Konzept der Kapselung erreicht, die einerseits das Verbergen von Daten (*Data Hiding*), viel wichtiger jedoch, das Verbergen von Informationen (*Information Hiding*) umfasst [28]. Das von Parnas [218, 219] eingeführte Prinzip des »Verbergens von Informationen« sieht vor, alle Informationen eines Moduls, die nicht zur essentiellen Charakteristik beitragen, zu verhüllen. Auf diese Weise wird erreicht, dass Module eines komplexen Systems nicht auf internen Details anderer Komponenten basieren [128] und so Änderungen am Programm in der Regel nur lokal und sicher wirksam werden [98].

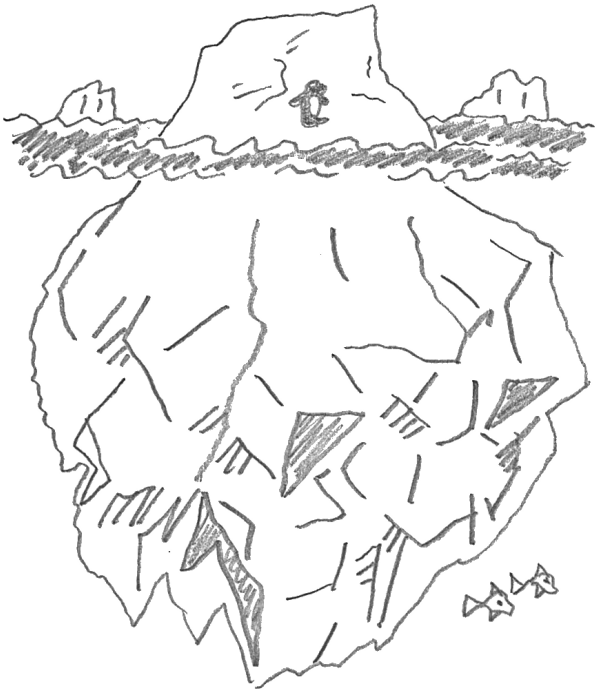


Abbildung 4.2: *Veranschaulichung des Verbergens von Informationen. Das Prinzip kann mit einem Eisberg verglichen werden, bei dem lediglich die Spitze (das öffentliche Interface) sichtbar ist, während ein Großteil (die Implementierung) unter der Wasseroberfläche verborgen bleibt (basierend auf [187]).*

Abstraktion

Während die Implementierung von internen Details im Vordergrund der Kapselung steht, liegt komplementär dazu der Fokus der Abstraktion auf dem von außen sichtbaren Verhalten. Abelson und Sussmann [1] nennen die Trennung von Verhalten und Implementierung eine Abstraktionsbarriere,

die durch das »Prinzip der geringsten Festlegung« (*Principle of least Commitment*) erreicht wird. Die Folge ist eine Reduktion der Schnittstelle eines Moduls auf das essentielle Verhalten.

Hierarchie

In den meisten nicht trivialen Softwaresystemen wird man in der Regel einer großen Anzahl an Abstraktionen begegnen. Um diese Menge verständlich zu halten, werden Abstraktionen allgemein in Gruppen zu Hierarchien zusammengefasst. Die Identifikation von Hierarchien innerhalb komplexer Systeme trägt dabei maßgeblich zum Verständnis bei.

4.2 Qualitativ hochwertiger Softwareentwurf

> »The gap between the best software engineering practice and the average practice is very wide — perhaps wider than in any other engineering discipline. A tool that disseminates good practice would be important.«
> Frederick P. Brooks

McConnell [187] weist darauf hin, dass Softwareentwicklung kein deterministischer Prozess ist. So existiert keine Wunderwaffe (»No Silver Bullet« [41]), die unfehlbar von den Anforderungen zu einer Implementierung eines komplexen Softwaresystems führt, das zudem einer hohen Qualität entspricht und somit leicht wartbar, erweiterbar und wiederverwendbar ist. Der Entwurf von Software ist vielmehr ein evolutionärer Prozess, der ein inkrementelles und iteratives Vorgehen beinhaltet. Darüber hinaus erlauben die Elemente des objektorientierten Softwareentwurfs ein äußerstes Maß an Flexibilität, die nahezu jede Form der Abstraktion erlaubt. Die Entscheidung für eine bestimmte Modularisierung ist dabei eine ebenso große Herausforderung wie die Wahl der Abstraktionen [28]. Während jedoch in traditionellen Ingenieurdisziplinen wie beispielsweise dem Bauwesen einheitliche Normen und Standards die Qualität der Konstruktion sicherstellen, existieren nur wenige solcher Standards im relativ jungen Bereich der Softwareentwicklung. Der Softwarentwurf, dessen Qualität maßgeblich durch die versierte Anwendung von Prinzipien (*Principles*) und Mustern (*Patterns*) bestimmt wird, ist vielmehr heuristischer Natur.

4.2.1 Heuristiken und Prinzipien

Basierend auf den gesammelten Erfahrungen im Bereich der Softwareentwicklung haben sich im Laufe der Zeit eine Zahl an Praktiken und Prinzipien herausgebildet, die zu einem qualitativen Entwurf beitragen. Diese heuristischen Prinzipien dienen als Richtlinien für häufige Problemstellungen. Sie bieten damit eine wertvolle Hilfestellung bei Entwurfsentscheidungen. Als Heuristiken sind die Prinzipien empirischer Natur. Obwohl sie sich in zahlreichen Situationen bewährt haben, können sie nicht als allgemeingültig angesehen werden. Keinesfalls sollten die Prinzipien als Gesetze angesehen werden, denen dogmatisch Folge zu leisten ist.

Als Beispiele für die hier betrachteten Prinzipien können unter anderem die bereits erwähnten Konzepte »Trennung der Anliegen« (SoC) und das »Verbergen von Informationen« angeführt werden. Als wohl bekannteste Prinzipien der Objektorientierung sollen an dieser Stelle die von Robert C. Martin

[179] zusammengetragenen »*SOLID-Principles*« kurz erleutert werden. Eine detailiertere Betrachtung kann unter anderem [179, 180, 181, 182, 187, 234] entnommen werden.

Das Akronym »SOLID« steht in diesem Zusammenhang stellvertretend für folgende Prinzipien:

- ***S**ingle Responsibility Principle* (SRP)
- ***O**pen Closed Principle* (OCP)
- ***L**iskov Substitution Principle* (LSP)
- ***I**nterface Segregation Principle* (ISP)
- ***D**ependency Inversion Principle* (DIP)

Single Responsibility Principle (SRP)

Ein Modul sollte nur aus einem einzigen Grund geändert werden müssen.

Das »Prinzip der einzigen Verantwortung« wurde von Robert C. Martin [179] eingeführt und basiert auf dem Prinzip der Kohäsion, beschrieben von Tom DeMarco [70]. Es besagt, dass ein Modul nur eine einzige Verantwortung besitzen sollte, wobei diese Verantwortung mit einem Änderungsgrund gleichzusetzen ist.

Open Closed Principle (OCP)

Module sollten offen (für Erweiterungen) und geschlossen (für Änderungen) sein.

Bertrand Meyer [191] hat das »Offen-Geschlossen Prinzip« definiert. Es besagt, dass Softwareeinheiten Möglichkeiten anbieten sollten, ihr Verhalten zu ändern oder zu erweitern, ohne das Modul selbst ändern zu müssen. Möglich wird dies durch das Schlüsselelement der Abstraktion, mit dessen Hilfe die Module um Schnittstellen erweitert werden können. Diese ermöglichen es, Funktionalität bei Bedarf zu ändern und zu ergänzen, ohne Änderungen an dem betreffenden Modul zu erfordern.

Liskov Substitution Principle (LSP)

Subtypen müssen durch ihre Basistypen ersetzt werden können.

Das »Liskovsche Substitutionsprinzip« oder »Ersetzbarkeitsprinzip« wurde von Barbara Liskov [168] formuliert. Es fordert, dass eine Instanz eines Subtyps sich so verhalten muss, dass jemand, der meint, ein Objekt des Basistyps vor sich zu haben, nicht durch unerwartetes Verhalten überrascht wird, wenn es sich dabei tatsächlich um ein Objekt des Subtyps handelt.

Interface Segregation Principle (ISP)

Aufrufer sollten nicht dazu gezwungen sein, von Schnittstellen abzuhängen, die sie nicht verwenden.

Das »Prinzip der Aufteilung von Schnittstellen« wurde ebenfalls von Robert C. Martin [178] eingeführt und fordert dazu auf, feingranulare Schnittstellen zu entwerfen, die exakt den Anforderungen der Aufrufer genügen.

Dependency Inversion Principle (DIP)

A) Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen.
B) Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen.

Das »Prinzip der Umkehrung von Abhängigkeiten« wurde gleichfalls von Robert C. Martin [177] beschrieben. Es fordert, dass Abhängigkeiten immer von konkreten Modulen niedriger Ebene in Richtung abstrakter Module höherer Ebene gerichtet sein sollten.

Weitere Prinzipien

Neben den hier erwähnten Prinzipien existieren noch zahlreiche weitere, nicht weniger bedeutsame Richtlinien für den objektorientierten Entwurf. Dazu gehören unter anderem:

- Law of Demeter (LoD) → Lieberherr [162]
- Don´t Repeat Yourself (DRY) → Hunt und Thomas [127]
- Keep it simple, stupid (KISS) → Booch [28]
- Favour Composition over Inheritance (FCoI) → Gamma et al. [97]

Abschließend sei an dieser Stelle gesagt, dass die Anwendung der Prinzipien nicht notwendigerweise zu einem qualitativen Entwurf führt. Martin [180] gibt zu bedenken, dass es für einen guten Softwareentwurf mehr bedarf als lediglich der Anwendung von Prinzipien und Mustern. Dazu gehören auch Aufmerksamkeit, Disziplin und die Leidenschaft, etwas Elegantes zu erschaffen.

4.2.2 Muster

Die vorausgehend erläuterten objektorientierten Prinzipien spiegeln sich unter anderem in sogenannten Mustern [97] wider. Sie stellen die erfolgreiche Anwendung der Prinzipien auf spezifische, wiederkehrende Entwurfsprobleme dar, indem sie in abstrakter Form generische Ansätze flexibler, eleganter und wiederverwendbarer Lösungen beschreiben. Nach Buschmann et al. [50] lassen sich die schablonenartigen Lösungen dabei hinsichtlich ihres Abstraktionsgrades nach Architekturmustern, Entwurfsmustern und Idiomen kategorisieren (vgl. Abbildung 4.3). Muster können somit auf nahezu jeder Ebene des Softwareentwurfs Anwendung finden.

Auf oberstem Abstraktionsniveau beschreiben Architekturmuster grundlegende Strukturprinzipien von Softwaresystemen, auf Basis vordefinierter Subsysteme mit spezifizierten Zuständigkeitsbereichen und festgelegten Beziehungen. Beispiele bekannter Architekturmuster sind das in Abschnitt 4.3.1 beschriebene *Layer*-Muster und das für klassische grafische Benutzeroberflächen eingesetzte Muster *Model View Controller* (MVC) (vgl. Abschnitt 4.3.2). Die Subsysteme und deren Beziehungen untereinander bestehen in der Regel aus kleineren architektonischen Einheiten, die mit Hilfe von Entwurfsmustern beschrieben werden können. Sie stellen Lösungsansätze für wiederkehrende Strukturen miteinander kommunizierender Komponenten in einem bestimmten Kontext dar.

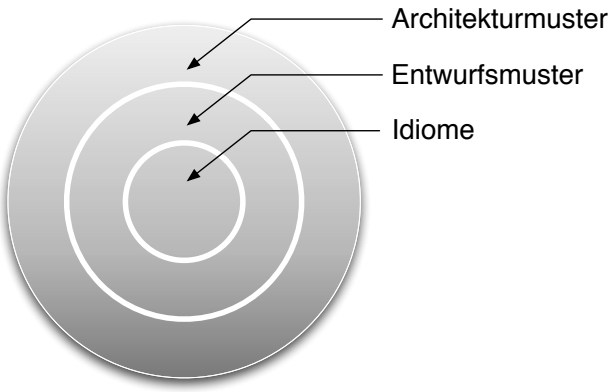


Abbildung 4.3: *Kategorisierung von Mustern hinsichtlich ihres Abstraktionsgrads nach Buschmann et al. [50].*

Ein Beispiel für Entwurfsmuster ist das in Abschnitt 4.3.5 erläuterte *Command*-Muster. Idiome hingegen sind spezifische Muster auf niedriger Abstraktionsebene, die sich auf Entwurfs- und Implementierungsaspekte in einer bestimmten Programmiersprache beziehen. Ein bekanntes Beispiel ist das *Counted-Pointer*-Idiom, das Referenzzähler zur Verwaltung von dynamisch allokiertem Speicher verwendet [50].

Das Konzept der Muster entstammt ursprünglich dem Bauwesen. Der Architekt Christopher Alexander geht in seinem Werk »The Timeless Way of Building« der Frage nach objektivierbarer Qualität von Bauwerksarchitekturen nach [6]. Diese Konzepte haben unter anderem Gamma et al. [97] auf die Softwareentwicklung übertragen und schufen damit eine bahnbrechende Referenz für den Entwurf komplexer Softwaresysteme. Die beschriebenen Muster erläutern den Entwurf sowie die Entwurfsentscheidungen einiger der besten und am häufigsten wiederverwendeten Softwaresysteme [48] und bieten damit praktische, hoch qualitative Lösungen für wiederkehrende Entwurfsprobleme an. Neben einer Steigerung der Qualität einer Lösung kann der Einsatz von Mustern zusätzlich zu einem deutlichen Produktivitätsgewinn führen, da das Erarbeiten einer eigenen Lösung zu Gunsten eines existierenden Ansatzes entfällt. Muster vermeiden es demzufolge, das sprichwörtliche Rad neu zu erfinden. Darüber hinaus legt das Werk von Gamma et al. den Grundstein für ein einheitliches Vokabular und fördert damit eine effiziente Kommunikation sowohl in Diskussion als auch Dokumentation. Die Verwendung eines Musters lässt dabei unmittelbar auf die Rolle, Beziehungen und Limitierungen einer Komponente innerhalb des Entwurfs rückschließen und trägt damit erheblich zum Verständnis der Lösung bei. Die genannten Vorteile von Entwurfsmustern wurden beispielsweise von Tichy durch Studien bestätigt [276].

4.2.3 Ziele und Metriken

Bauingenieure kämen wahrscheinlich nicht auf die Idee, unterhalb eines existierenden Hochbaus mit etwa 100 Ebenen ein weiteres Stockwerk einzuziehen, da ein solches Vorhaben nicht zuletzt mit immensen Kosten verbunden wäre und zudem eine erhebliche Fehlerträchtigkeit aufwiese. Vorhaben dieser Art würden sicher für Furore sorgen. Übertragen auf die Softwareentwicklung sind Forderungen nach derartigen Maßnahmen jedoch keine Seltenheit.

Zusätzlich zu den inkrementellen und iterativen Eigenschaften des Softwareentwicklungsprozesses erfordern häufig wechselnde Anforderungen kontinuierliche Änderungen und Erweiterungen der Systeme. Der Hauptgrund, die interne Qualität der Anwendung sicherzustellen, besteht daher darin, sichere und vorhersehbare Modifikationen des Systemverhaltens zu realisieren, ohne dabei tiefgreifende Änderungen am Entwurf vornehmen zu müssen [93]. Basierend auf einer Studie [106] besitzen kompetente Softwareentwickler insbesondere die Fähigkeit Änderungen vorherzusehen. Diesen Änderungen zuvorzukommen stellt einen der wichtigsten Aspekte des Entwurfs dar. Das Ziel dabei ist, die volatilen Bereiche eines Softwaresystems zu identifizieren und von den nicht volatilen Bereichen zu isolieren, um die Auswirkungen von Modifikationen einzugrenzen [187]. Ingalls [129] schlägt vor, den Kern eines System aus einer geringen Anzahl abstrakter, unveränderlicher Komponenten zu erstellen. Diese sollten so generell wie möglich sein und die Grundfunktionalität eines einheitlichen Frameworks bereitstellen. Konkrete Spezialisierungen hingegen können das Framework um zusätzliche Funktionalität erweitern.

Zur Beurteilung der Qualität des Entwurfs im Hinblick auf die Auswirkungen und Reichweite von Modifikationen können unter anderem die von Yourdon und Constantine [317] vorgeschlagenen Metriken (Kopplung und Kohäsion) herangezogen werden. Obwohl diese Kriterien ursprünglich der strukturellen Programmierung entstammen, lassen sie sich informell auch auf die Objektorientierung übertragen. Dabei ist die Kopplung ein Maß für die Stärke der Bindung zweier Module [264]. Stehen diese in enger Wechselbeziehung, werden die Module als stark gekoppelt bezeichnet. Eine starke Kopplung führt (im Gegensatz zu einer losen Kopplung) zu Systemen, die einerseits schwer verständlich, vor allem aber schwer anzupassen und zu korrigieren sind, da Änderungen an einem Modul ebenfalls Änderungen an den gekoppelten Modulen erforderlich machen. Demgegenüber misst die Kohäsion den Grad der Zusammengehörigkeit der Elemente innerhalb eines Moduls [264]. Dabei entspricht die zufällige Kohäsion, bei der nicht zusammengehörige Abstraktionen innerhalb eines Moduls gruppiert werden, der geringsten und damit am wenigsten gewünschten Form der Kohäsion. Im Gegensatz dazu ist eine hohe funktionale Kohäsion wünschenswert, bei der die Summe der Elemente eines Moduls ein einheitliches, klar definiertes Verhalten formt. Im Fall einer hohen Kohäsion sind Änderungen hinsichtlich eines bestimmten Verhaltens deutlich einfacher umzusetzen, da sie sich im Idealfall auf ein einzelnes Modul reduzieren lassen. Das Ziel des Entwurfs sollte also stets eine möglichst hohe Kohäsion bei gleichzeitig loser Kopplung sein.

Zusammengefasst lassen sich durch einen hochwertigen, objektorientierten Entwurf Anwendungen erstellen, die sich robust gegenüber Änderungen verhalten. Durch eine intelligente Zerlegung des Zustandsraumes wird dabei ein hohes Maß an Sicherheit hinsichtlich der Korrektheit der Lösung erzielt und dadurch schlussendlich das Versagensrisiko von Software reduziert.

4.3 Basisentwurf von Virtual Fluids Interactive

> »It is easier to write ten volumes on theoretical
> principles than to put one into practice.«
> Lew Nikolajewitsch Tolstoi

Die meisten Anwendungen im wissenschaftlichen Umfeld entstehen isoliert. Sie werden im Allgemeinen von ein und demselben Entwickler entworfen, implementiert, gewartet und auch verwendet.

Häufig handelt es sich dabei um Spezialanwendungen mit eng umfasster Funktionalität für detaillierte Problemstellungen und prototypartiger Implementierung. Auf Grund des eingeschränkten, projektbezogenen Anwendungsbereichs ist die Lebensdauer dieser Applikationen jedoch in der Regel eher begrenzt. Nach Booch [28] werden derartige Anwendungen daher als nicht komplex kategorisiert. Das bedeutet nicht, dass diese Anwendungen weniger anspruchsvoll sind. Zweifelsohne handelt es sich häufig um wohl überlegte, intelligente Lösungen. Dennoch werden die sehr speziellen Anwendungen in neuen Projekten häufig zu Gunsten einer Neuimplementierung verworfen, anstatt sie wiederzuverwenden[2], zu modifizieren oder ihre Funktionalität zu erweitern.

Für VIRTUALFLUIDS INTERACTIVE wird hingegen ein nachhaltiger Ansatz hoher interner Qualität angestrebt, der eine flexible, wartbare und einfach zu erweiternde Anwendung ermöglicht und auf wechselnde Anforderungen vorbereitet ist. Darüber hinaus ist VIRTUALFLUIDS INTERACTIVE nicht nur eine alleinstehende Applikation sondern stellt eine Rahmenumgebung (*Framework*) zur Verfügung, die zur Lösung spezifischer Strömungsprobleme erweitert werden kann. Auf diese Weise sollen anderen Entwicklern die Vorteile von *Computational Steering* (vgl. Abschnitt 2.2.3) zugänglich gemacht werden. VIRTUALFLUIDS INTERACTIVE besteht daher aus einem abstrakten Applikationskern, basierend auf kohärenten Strukturen. Dazu werden nach Möglichkeit gängige Muster eingesetzt, die sich ebenfalls in der Namensgebung widerspiegeln und zur Verständlichkeit beitragen. Der beständige Anwendungskern zeichnet sich insbesondere durch Abstraktionen aus, die Schnittstellen für konkrete Implementierungen definieren. Diese ermöglichen es, das existierende Verhalten für zusätzliche Problemstellungen zu erweitern.

4.3.1 Schichtenarchitektur

Wie Alan Kay, der Erfinder der Objektorientierung betont, liegt der Fokus des objektorientierten Entwurfs weniger auf den Objekten selbst, als vielmehr auf der Kommunikation zwischen den Objekten [139]. Im Folgenden werden daher zunächst die modularen Strukturen und deren Interaktionen auf hohem Abstraktionsniveau betrachtet. Zu diesem Zweck werden erforderliche Schlüsselabstraktionen identifiziert und in den globalen Kontext eingeordnet.

Zu Beginn wird dazu die logische Architektur, das heißt, die globale Struktur der Anwendung, betrachtet. Auf oberster Ebene erfolgt eine Organisation komplexer Softwaresysteme im Allgemeinen durch eine Einteilung in Schichten. Eine Schicht entspricht dabei einer grobgranularen Gruppierung von Subsystemen, die zusammen für einen Hauptaspekt des Systems verantwortlich sind [157]. Bei dieser Einteilung kommt das sogenannte *Layer*-Architekturmuster zur Anwendung [50]. Interaktive Systeme werden an dieser Stelle in der Regel untergliedert in eine Präsentationsschicht (*Presentation Layer*), Domänenschicht (*Domain Layer*) und Datenschicht (*Data Layer*) [91] (vgl. Abbildung 4.4).

Generell setzen dabei höhere Anwendungsebenen auf niedrigeren Schichten auf, wobei die unteren Schichten wiederum keine Kenntnisse von höheren Ebenen besitzen. Des Weiteren verbirgt jede Ebene in der Regel alle tiefer liegenden Schichten vor höheren Schichten. Erreicht wird eine derartige Untergliederung des Softwaresystems durch die Anwendung der bereits diskutierten Entwurfsprinzipien wie beispielsweise *Separation of Concerns*, *Open Closed* und *Dependency Inversion*. Das Ergebnis

[2] Auf algorithmischer Ebene bzw. (niedriger) Implementierungsebene findet durchaus eine Wiederverwendung statt. An dieser Stelle ist jedoch eher die Entwurfsebene gemeint.

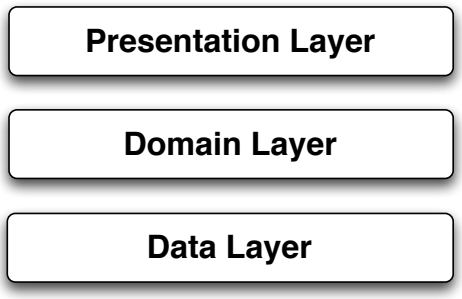


Abbildung 4.4: *Ebenen einer typischen interaktiven Softwarearchitektur.*

ist ein System hoher Orthogonalität aus Anwendungsschichten mit geringer Kopplung und hoher *Kohäsion*, wodurch sich unter anderem folgende Vorteile ergeben:

- Anwendungsebenen sind auch ohne Kenntnis der übrigen Schichten für sich genommen verständlich.
- Einzelne Schichten können durch alternative Implementierungen ersetzt werden, ohne andere Schichten zu beeinflussen.
- Abhängigkeiten zwischen verschiedenen Schichten werden minimiert.

Die Präsentationsschicht bildet allgemein die Schnittstelle zwischen dem Anwender und der Applikation. In VIRTUALFLUIDS INTERACTIVE wird diese Aufgabe von drei unterschiedlichen Komponenten übernommen (vgl. Abbildung 4.5). Die grafische Benutzeroberfläche enthält dabei sowohl klassische Steuerelemente als auch Elemente für die Darstellung und Interaktion mit dreidimensionalen Geometrie- und Ergebnisdaten, für die jeweils eine separate Komponente existiert. Die UI-Komponente zeigt dabei klassische Formularansichten (*Views*) an, während die *Rendering*-Komponente die Darstellung dreidimensionaler »Grafikobjekte« (*Graphics*) übernimmt. Die Schnittstellen `View` und `Graphic` bilden damit die Schlüsselabstraktionen dieser Anwendungsebene. Die dritte Komponente der Präsentationsschicht übernimmt die Steuerung und Kontrolle der sowohl klassischen als auch dreidimensionalen Darstellung. Grafikobjekte und deren Steuerung werden in Abschnitt 6.4.4 behandelt.

Die Präsentationsebene basiert auf der Domänenschicht, welche die eigentliche Applikationslogik und das Domänenmodell umfasst. Dazu zählt eine Infrastruktur zur Verwaltung des Simulationsprojektes, das im Fall von VIRTUALFLUIDS INTERACTIVE eine Baumstruktur bildet. Ein Element dieser Projekthierarchie ist dabei stets vom Typ `HierarchyNode`. Hierarchieelemente können beispielsweise Geometrien sein, die im Kontext der Modellierungskomponente ein geometrisches Modell als Ausgangspunkt für eine Simulation formen. Zu diesem Zweck unterstützt die Komponente die Generierung von uniformen Berechnungsgittern auf Basis von Geometrieobjekten. Die Abstraktionen `UniformGrid` und `Geometry` stellen damit ein immanentes Element der Komponentenschnittstelle dar. Als diskrete Geometriebeschreibung bilden die uniformen Berechnungsgitter neben einem Parametersatz zur Definition von Strömungsgrößen die Eingangsdaten für die Simulationskomponente. Diese wiederum offeriert Ergebnisdaten in Form der Abstraktion `GridData`. Eine detaillierte Beschreibung der Simulationskomponente erfolgt in Abschnitt 5.4.

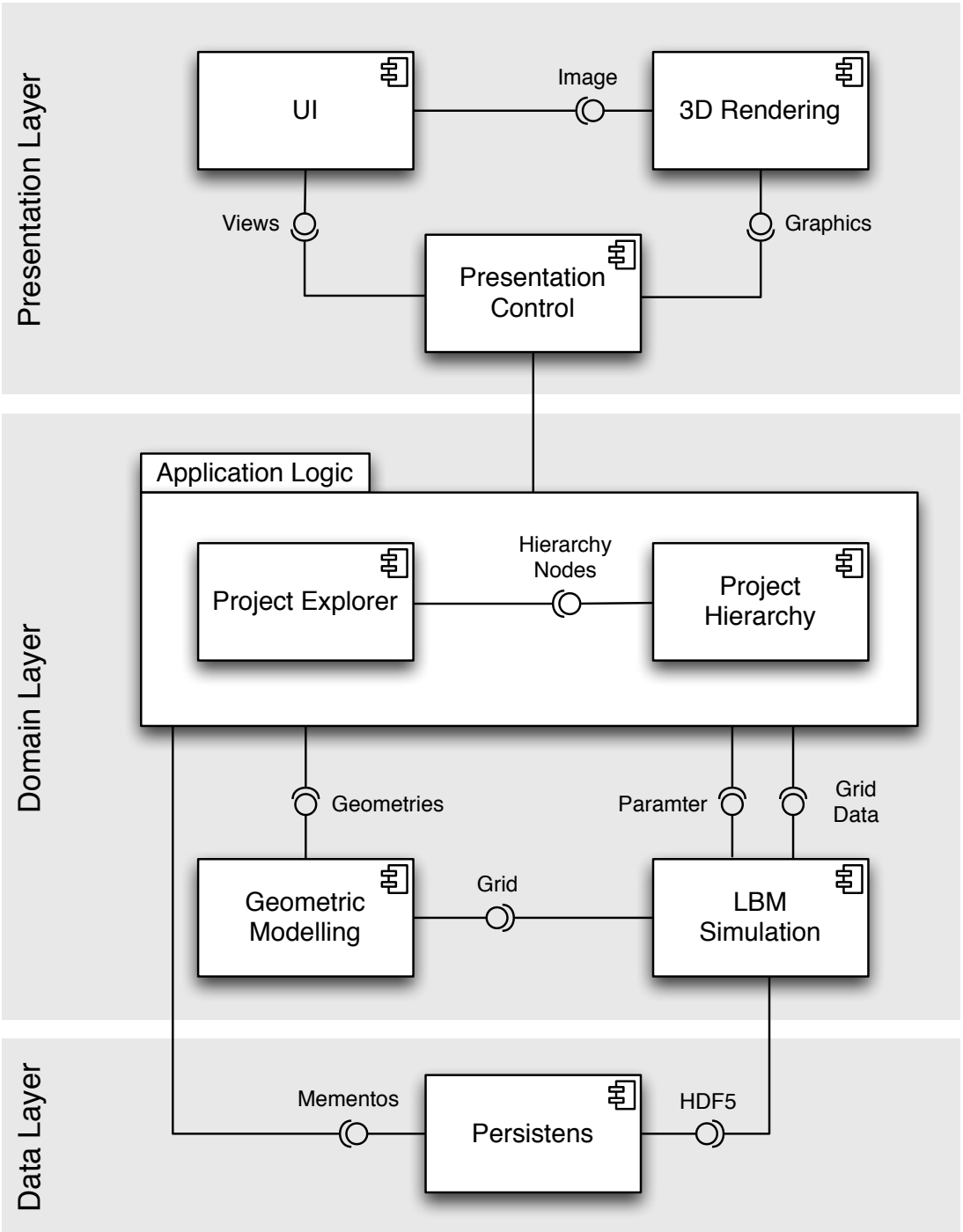


Abbildung 4.5: *Veranschaulichung der Subsysteme innerhalb von* VIRTUALFLUIDS INTERACTIVE.

Sowohl die Modellierungskomponente als auch die Simulationskomponente machen Gebrauch von der Datenschicht. Diese Anwendungsebene dient im Allgemeinen dazu, Daten persistent zu speichern. Die Persistenzkomponente bietet daher Schnittstellen zum Speichern und Laden von Geometriemodellen wie auch von Ergebnisdaten aus Simulationen. Ergebnisdaten werden in VIRTUALFLUIDS INTERACTIVE im HDF5-Format gespeichert und können anschließend in einem späteren *Post-Processing*-Schritt erneut analysiert werden. Dieser Themenkomplex wird detailliert in Abschnitt 6.5.3 behandelt.

4.3.2 Organisation der Präsentationslogik

Als Schnittstelle zwischen dem Anwender und der Applikation nimmt die Präsentationsschicht eine besondere Rolle ein. Die grafische Benutzeroberfläche ist oft von entscheidender Bedeutung für den Erfolg oder Misserfolg einer Anwendung, weshalb diesem Aspekt im Rahmen dieser Arbeit besondere Aufmerksamkeit gewidmet wird. Die Organisation der Präsentationsschicht in VIRTUALFLUIDS INTERACTIVE ist angelehnt an das bekannte Architekturmuster *Model View Controller* (MVC). Dabei handelt es sich um ein traditionelles Muster zur Gestaltung interaktiver grafischer Benutzerschnittstellen, das der Programmiersprache Smalltalk-80 [108] entstammt und erstmals von Reenskaug beschrieben wird [232]. Bei dem Entwurf heutiger komponentenbasierter Benutzerschnittstellen kommt der klassische MVC-Ansatz in seiner ursprünglichen Art jedoch kaum noch zur Anwendung. Stattdessen werden heute im Allgemeinen Variationen des Musters eingesetzt. Ein Nachfolger ist das *Model View Presenter*-Muster (MVP), das in seiner ursprünglichen Form von Potel [225] beschrieben wird und durch Dolphin Smalltalk [31] weitere Verbreitung findet. Ebenso wie das klassische *Model View Controller*-Muster adressiert MVP in erster Linie eine klare »Trennung der Anliegen« (SoC). Eine Gemeinsamkeit aller Präsentationsmuster ist das Konzept der Trennung von *Model* und *View* (*Model/View Separation*) [157]. Der Grundgedanke dabei ist, die Präsentationslogik von der Domänenlogik zu separieren, anstatt die Komplexität von Domänenobjekten durch Präsentationslogik zu erhöhen und damit das »Prinzip der einzigen Verantwortung« (SRP) zu verletzten. Im Fall des *Model View Presenter*-Musters erfolgt die Trennung der Verantwortungen hinsichtlich Domänenobjekte sowie der Darstellung dieser Daten und der dazu erforderlichen Präsentationslogik. Fowler untergliedert das *Model View Presenter*-Muster weiterhin in die Variationen *Supervising Controller* und *Passive View* [92].

Im Rahmen dieser Arbeit wird der *Passive View*-Variation der Vorzug gegeben, da sie verglichen mit dem *Supervising Controller*-Ansatz sehr explizit ist [92]. Die Struktur dieses Musters ist in Abbildung 4.6 veranschaulicht. Eine *View*-Komponente entspricht in diesem Fall einem Fenster oder Formular, bestehend aus mehreren Steuerelementen. In VIRTUALFLUIDS INTERACTIVE kann dies beispielsweise das Hauptanwendungsfenster, ein Formular zur Darstellung und Manipulation von Parametern oder auch ein Dialogfenster sein (vgl. Abbildung 4.7). *View*-Objekte entsprechen dabei einer visuellen Repräsentation von *Model*-Objekten aus der Domänenschicht. Im Allgemeinen existiert für jedes grafisch darzustellende Domänenobjekt analog eine entsprechende MVP-Triade mit spezifischen Implementierungen von *View* und *Presenter*. *Presenter*-Objekte übernehmen dabei die Rolle von »Vermittlern« zwischen *View*- und *Model*-Objekten. In der *Passive View*-Variation implementieren sie die gesamte Darstellungslogik, während sich die äußerst schlanken *View*-Objekte vollständig passiv verhalten, da sie keine Kenntnis von den *Model*-Objekten besitzen. Ihre Logik reduziert sich

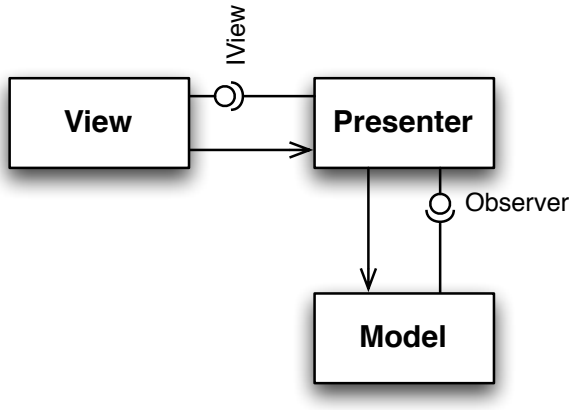


Abbildung 4.6: *Passive View-Variante des Model View Presenter-Musters (MVP).*

auf die initiale Behandlung von Ereignissen, ausgelöst durch Benutzerinteraktionen mit den Steuerelementen. Die Kontrolle über die Verarbeitung wird jedoch unmittelbar an die *Presenter*-Objekte weitergereicht, von denen letztlich die Aktualisierung der jeweiligen Domänenobjekte vorgenommen wird. In VIRTUALFLUIDS INTERACTIVE geschieht die Synchronisierung dabei mit Ausnahme von Dialogen auf Ebene der einzelnen Steuerelemente eines Formulars (*Field Synchronisation* [92]). In diesem Fall werden Domänenobjekte instantan nach Interaktion des Benutzers mit einem Steuerelement aktualisiert. Im Gegensatz dazu erfolgt die Aktualisierung bei Dialogen auf Basis des gesamten Formulars nach erfolgreichem Abschluss des Dialogs (*Screen Synchronisation* [92]).

In entgegengesetzter Richtung werden *Presenter* über Änderungen an den *Model*-Objekten mit Hilfe der *Observer*-Musters [97] (Beobachtermuster) benachrichtigt und aktualisieren daraufhin die spezifische Darstellung. Um über Änderungen informiert zu werden, implementieren *Presenter*-Objekte eine *Observer*-Schnittstelle. Der *Observer*-Mechanismus wird in VIRTUALFLUIDS INTERACTIVE mit Hilfe des Signal-Verfahrens aus der Boost-Bibliothek [138] umgesetzt.

Das Ergebnis ist ein flexibler, kohäsiver Entwurf aus robusten Komponenten mit klar abgegrenzten Anliegen (SoC) und Verantwortungen (SRP), der deutliche Vorteile mit sich bringt. So ist es möglich, die *Model*-, *View*- und *Presenter*-Komponenten im Wesentlichen unabhängig voneinander zu implementieren und zu verändern, ohne andere Bereiche der Triade zu beeinflussen. Das Layout eines Fensters oder Formulars kann beispielsweise angepasst werden, ohne Änderungen an der Präsentationslogik nach sich zu ziehen. Weiterhin ist es möglich, das Verhalten eines Formulars zur Laufzeit durch Austausch des jeweiligen *Presenter*-Objekts dynamisch zu variieren. Auf diese Weise werden Fenster beispielsweise deaktiviert oder bekommen den Schreibzugriff auf einzelne Steuerelemente entzogen. Damit verhalten sich die *Presenter*-Objekte analog zu Strategieobjekten im Kontext des sogenannten Strategiemusters (*Strategy Pattern*) [97]. Ein wesentlicher Vorteil des *Model View Presenter*-Musters ist seine Eignung für Tests. Generell sind Tests der Benutzerschnittstellen nur sehr eingeschränkt möglich [92, 182]. Der Vorteil des MVP-basierten Entwurfs liegt in einer extrem reduzierten Logik der *View*-Komponenten, wodurch das Fehlerrisiko in der Darstellungsschicht minimiert wird. Den Großteil der Anwendungslogik implementieren hingegen die erheblich einfacher zu testenden *Presenter*- und *Model*-Klassen.

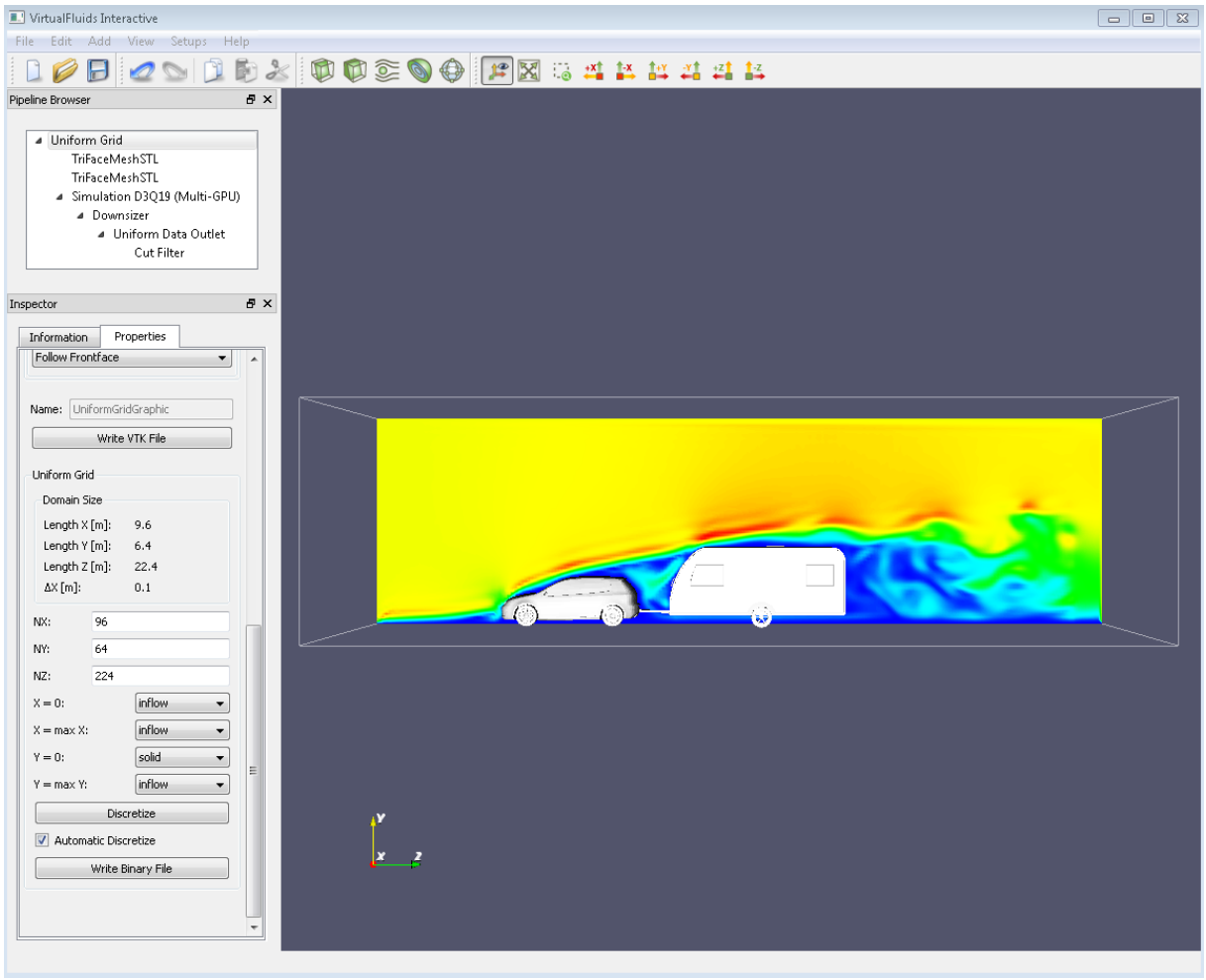


Abbildung 4.7: *Das Hauptanwendungsfenster von* VIRTUALFLUIDS INTERACTIVE *umfasst drei wesentliche Elemente: das Darstellungsfenster der 3D-Visualisierung (rechts), die Projekthierarchie (links oben) und das kontextabhängige Eigenschaftenfenster (links unten).*

Weiterhin lässt sich die Abhängigkeit der *Presenter*-Objekte von den *View*-Implementierungen durch eine zusätzliche Schnittstelle (`IView`) invertieren (DIP). Dadurch wird einerseits die Testbarkeit der Anwendung gesteigert und andererseits eine Möglichkeit gegeben, alternative *View*-Klassen anzubieten, die beispielsweise auf einem anderen *UI-Framework* basieren. Das derzeitige Benutzerinterface von VIRTUALFLUIDS INTERACTIVE basiert auf dem Qt-Framework [268], das unter Linux und Windows ein (nahezu) natives Erscheinungsbild und Interaktionsverhalten ermöglicht, unter OS X jedoch faktisch kein zufriedenstellendes Ergebnis erzielt. Denkbar wären an dieser Stelle beispielsweise Implementierungen auf Basis von Cocoa [122] unter OS X oder Windows Forms [250] und WPF [249] auf der Windows Plattform, um die Vertrautheit des Anwenders mit der Benutzerschnittstelle weiter zu steigern. Derweil erscheint die Qt-Bibliothek als die beste Lösung, wenn gleichzeitig eine Plattformunabhängigkeit erwünscht ist.

Die Abhängigkeit der Rahmenapplikation von konkreten Implementierungen der *View*-Komponenten wird in VIRTUALFLUIDS INTERACTIVE mit Hilfe des *Abstract Factory*-Musters [97] aufgelöst (vgl. Abbildung 4.8). Die Verantwortlichkeit des Erstellens von *View*-Objekten obliegt

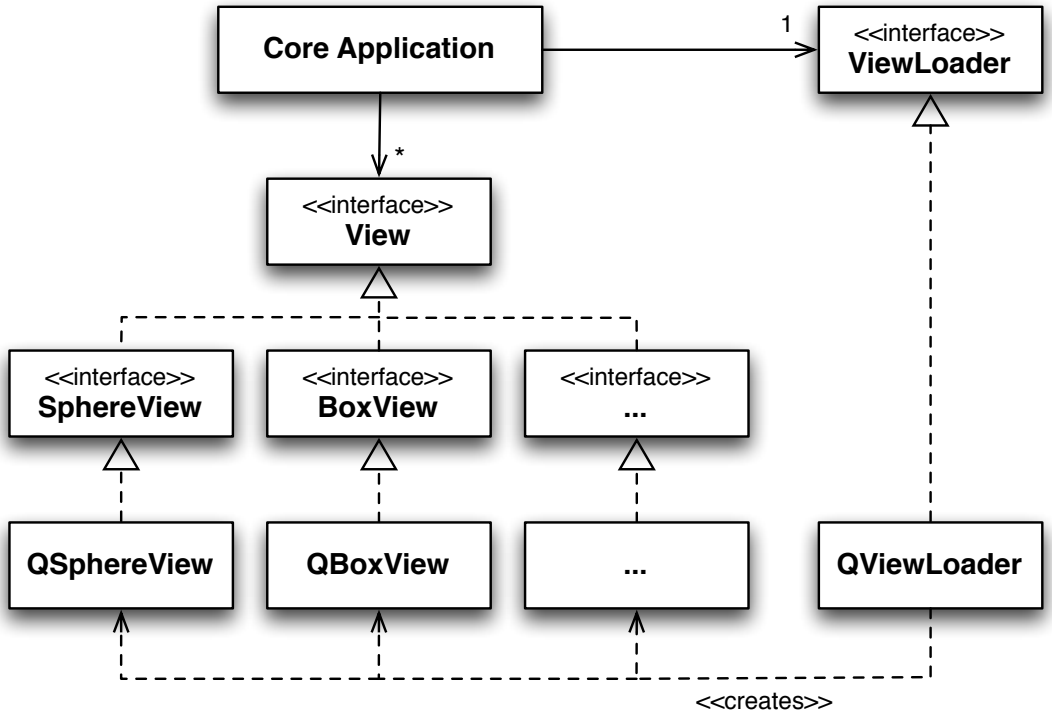


Abbildung 4.8: *Erstellen von View-Komponenten mit Hilfe des Abstract Factory-Musters.*

dabei einem »Fabrikobjekt« (SoC), das die Schnittstelle `ViewLoader` implementiert (DIP). Durch alternative Implementierungen des Fabrikobjekts kann so ein Wechsel der Benutzerschnittstelle erfolgen, ohne Änderungen an der Rahmenanwendung zu erfordern (OCP). Derzeit steht in VIRTUALFLUIDS INTERACTIVE mit der Implementierung der Klasse `QViewLoader` ein Fabrikobjekt auf Basis der Qt-Bibliothek zur Verfügung.

4.3.3 Projekthierarchie

VIRTUALFLUIDS INTERACTIVE verwaltet Projektdaten in Form einer baumartigen Hierarchie, vergleichbar einer typischen Verzeichnisstruktur von Datenträgern (vgl. Abbildung 4.9). Jedem Element

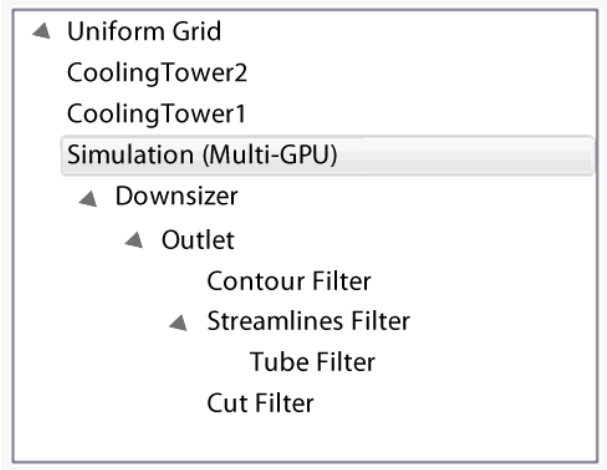


Abbildung 4.9: *Baumartige Hierarchiestruktur der Elemente einer Strömungssimulation.*

in der Hierarchie können Kindelemente zugeordnet werden, die in einer definierten Wechselbeziehung zu dem Elternknoten stehen. Seitens der Kindelemente wird diese Relation in der Regel in der Weiterverarbeitung von Daten der Elternelemente bestehen, so dass sich ausgehend von den Elternknoten ein gerichteter Datenstrom zu den Kindknoten einstellt. Eltern- und Kindknoten verhalten sich dabei analog zu dem *Pipes-and-Filters*-Muster [50]. Die Elemente der obersten Hierarchieebene stellen demnach Datenquellen (*Data Sources*) dar, die ihre Daten den Kindknoten (Filter) zur Weiterverarbeitung zugänglich machen. Datensenken bilden das Ende einer solchen, als Datenpipeline bezeichneten, Struktur. Sie realisieren beispielsweise die visuelle Darstellung im Kontext der 3D-Visualisierung. In VIRTUALFLUIDS INTERACTIVE können Kindelemente ihre Elternelemente zusätzlich um Informationen ergänzen. So werden Geometrieelemente in der Hierarchie beispielsweise als Kindknoten eines übergeordneten geometrischen Modells zur Beschreibung eines Strömungsgebiets gehalten. In diesem Fall erweitern die Kindknoten das Modell um zusätzliche Randbedingungen.

Die Hierarchiestruktur wird von der Klasse `ProjectHierarchy` repräsentiert, zu deren Verwaltung zusätzlich die Klasse `ProjectExplorer` existiert. Beide Kernkomponenten arbeiten grundsätzlich auf Basis der abstrakten Basisklasse `HierarchyNode`, von der eine gemeinsame Schnittstelle aller Hierarchieelemente definiert wird. Der Kern der Rahmenanwendung ist damit unabhängig von konkreten Implementierungen einzelner Elemente. Die Verantwortung des Erstellens konkreter Instanzen wird an ein Fabrikobjekt delegiert, das zusätzlich zu dem neuen Hierarchieelement ein Objekt vom Typ `HierarchyNodeBinder` zurückliefert. Mit Hilfe dieses Objekts wird die Bindung eines neuen Knotens an einen existierenden Elternknoten erreicht, obgleich zunächst keine Typinformationen über die jeweilige, konkrete Ausprägung des Knotens vorliegen.

Ermöglicht wird dies durch Einsatz des Besuchermusters (*Visitor*-Muster), das einen klassischen Ansatz zur Rückgewinnung von Typinformationen durch Implementieren eines *Double-Dispatch*-Mechanismus (doppelte Verteilung) darstellt [97]. Abbildung 4.10 veranschaulicht die Struktur des

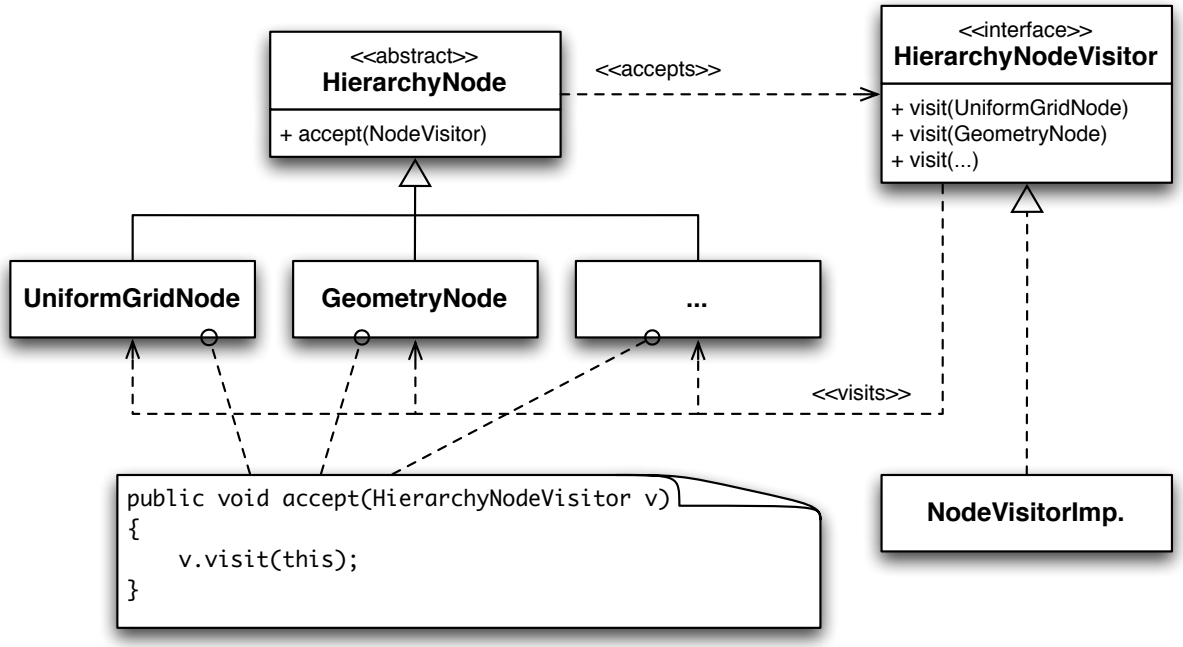


Abbildung 4.10: *Ursprüngliche Implementierung des Besuchermusters für die Projekthierarchie.*

klassischen Beobachtermusters[3] am Beispiel ausgewählter Elemente der Projekthierarchie. Für die Realisierung des *Double-Dispatch*-Mechanismus ist es erforderlich, dass alle Besucherimplementierungen die Schnittstelle `HierarchyNodeVisitor` umsetzen. Diese legt fest, dass jeder Besucher eines Hierarchieelements die Methode `visit` implementiert, die als Parameter den besuchten Knoten empfängt und für jeden existierenden Knotentyp überladen ist. Damit ein Hierarchieknoten besucht werden kann, muss dieser die Methode `accept` bereitstellen, die wiederum ein konkretes Besucherobjekt als Argument erwartet. Die einzige Aufgabe der `accept`-Methode besteht darin, die Ausführungskontrolle zu invertieren und unmittelbar an den Besucher durch Aufruf der `visit`-Methode zurückzugeben. Der konkrete Hierarchieknoten selbst wird dabei als Argument übergeben. Da sich dieser Methodenaufruf polymorph verhält, wird die jeweils überladene `visit`-Methode des Besucherobjekts ausgeführt und damit die Information über den Typ des besuchten Knotens zurückgewonnen.

Objekte des Typs `HierarchyNodeBinder` implementieren die Besucherschnittstelle und können somit beim Binden von Kind- an Elternknoten auf dem beschriebenen Schema aufsetzen. Zu jedem Kindknotentyp existiert analog eine entsprechende Spezialisierung der Klasse `HierarchyNodeBinder`, in der die notwendige Logik zum Binden des Kindknotens an mögliche Elternelemente implementiert ist (vgl. Abbildung 4.11). Jedes *Binder*-Objekt bekommt deshalb bei der Initialisierung einen Kindknoten zugewiesen, der bei Aufruf der Methode `bindTo` an ein anderes Hierarchieelement gebunden werden kann (vgl. Abbildung 4.12). Der Methodenaufruf er-

[3] Ein Nachteil der klassischen Implementierung des Besuchermusters besteht in zahlreichen Abhängigkeiten der beteiligten Klassen auf Quelltextebene. Aus diesem Grund wird eine Erweiterung der Rahmenanwendung um zusätzliche Hierarchieelemente erheblich erschwert. In VIRTUALFLUIDS INTERACTIVE wird die Mehrzahl dieser Abhängigkeiten durch die Implementierung der sogenannten *Acrylic Visitor*-Variante [183] aufgebrochen. Die grundsätzliche Struktur des Ansatzes bleibt davon jedoch unbeeinflusst.

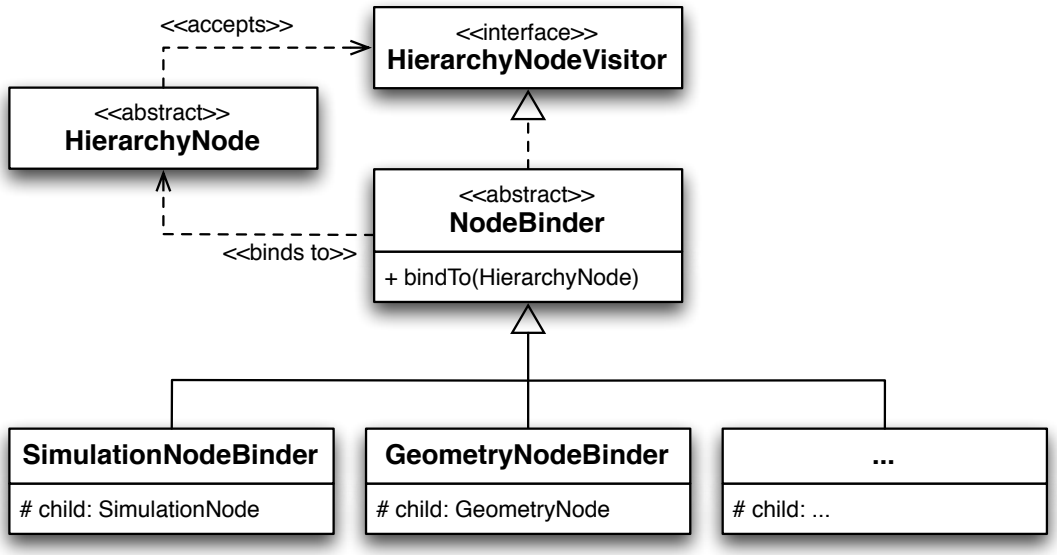


Abbildung 4.11: *Klassenhierarchie zur Bindung von Kind- an Elternknoten.*

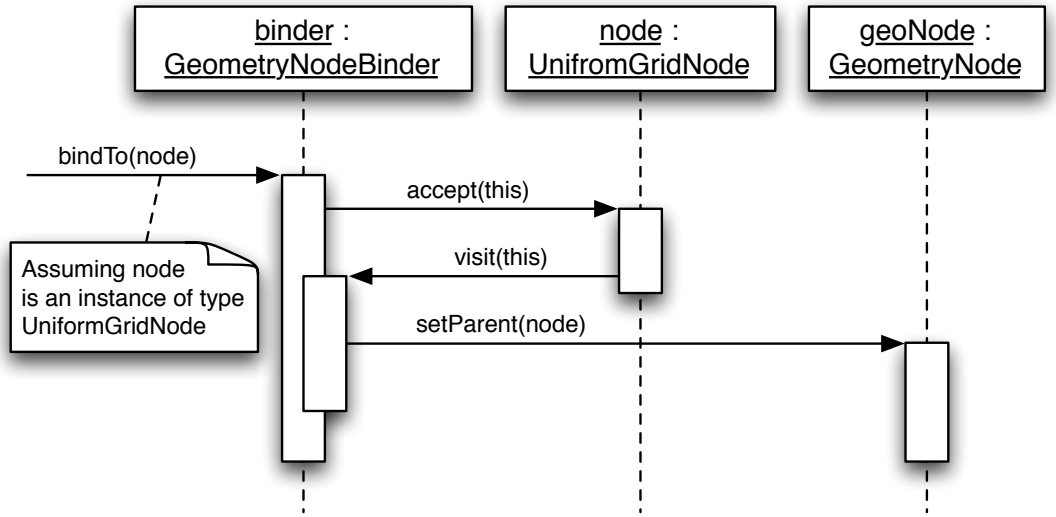


Abbildung 4.12: *Interaktion beteiligter Komponenten bei der Bindung von Kind- an Elternknoten.*

wartet grundsätzlich ein Elternelement vom Basistyp `HierarchyNode` und stößt anschließend den *Double-Dispatch*-Mechanismus an, um den Typ des Elternknotens zu bestimmen. Auf Basis dieser Information kann anschließend die Kompatibilität von Kind- und Elternknoten ermittelt werden und gegebenenfalls eine Bindung hergestellt werden. Geometrieobjekte können einer Projekthierarchie beispielsweise nur als Kindknoten eines Geometriemodells hinzugefügt werden. Zu diesem Zweck referenziert jedes Hierarchieelement die beteiligten Objekte der Domänenschicht und ist in der Lage die Beziehungen zwischen den Objekten herzustellen. Sind Kind- und Elternknoten hingegen inkompatibel, wird ein Ausnahmefehler ausgelöst.

Eine Erweiterung der Projekthierarchie um einen zusätzlichen Knotentyp würde demnach notwendigerweise die Implementierung einer weiteren Spezialisierung des Basistypens `HierarchyNode` erfordern sowie die Bereitstellung eines entsprechenden `HierarchyNodeBinder`-Besuchers für den Fall, dass neue Elemente als Kindknoten auf bereits vorhandenen Hierarchieelementen aufsetzen.

4.3.4 Kontextsensitive Präsentation

Das Besuchermuster findet ebenfalls in der Präsentationslogik Anwendung, um dort beispielsweise eine kontextsensitive Darstellung zu erzielen, die im Wesentlichen von dem derzeitig ausgewählten Element der Projekthierarchie abhängig ist. Die Klasse `ProjectHierarchy` setzt dazu das *Observer*-Muster um und informiert registrierte Beobachter auf diese Weise über die Selektion eines Elements. Einer dieser Beobachter ist ein Objekt vom Typ `ContextPresenterLocator`, dessen Aufgabe darin besteht, alle *Presenter*-Objekte zu ermitteln, die zur Repräsentation der Parameter des derzeit ausgewählten Hierarchieelements notwendig sind, so dass die entsprechenden Formulare anschließend in dem Kontext des dynamischen Eigenschaftenfensters zur Anzeige gebracht werden können (vgl. Abbildung 4.7). Die Umsetzung orientiert sich an dem von Fowler [92] beschriebenen *Service Locator*-Muster und setzt, wie bereits erwähnt, ebenfalls auf dem *Double-Dispatch*-Mechanismus der Hierarchieknoten auf.

Ähnliches gilt für die Steuerung der Menüleiste des Hauptanwendungsfensters, dessen Steuerung auf mehrere Presenterobjekte verteilt ist. Jede dieser Instanzen übernimmt dabei die Kontrolle eines der Hauptmenüpunkte, dessen Einträge in Abhängigkeit von dem selektiertem Hierarchieelement aktiviert oder deaktiviert werden.

4.3.5 Reversibilität

Sowohl über das Eigenschaftenfenster als auch die Menübefehle oder die direkte Manipulation mit der dreidimensionalen Darstellung wird der Anwender in die Lage versetzt, Änderungen an einzelnen Elementen der Projekthierarchie oder auch der Hierarchie selbst vorzunehmen. Zu diesem Zweck stellt unter anderem die Klasse `ProjectExplorer` diverse Funktionalitäten zur Verfügung. Die Operationen werden intern durch Anwendung des *Command*-Musters [97] von sogenannten Befehlsobjekten gekapselt. Das Muster erhebt damit einzelne Methoden auf die Ebene von Klassen und verstösst somit streng genommen gegen die objektorientierten Prinzipien, da es einer funktionalen Dekomposition gleichkommt. Auf diese Weise wird es jedoch möglich, eine Befehlshistorie anzulegen, auf deren Basis ein Mechanismus implementiert werden kann, der es erlaubt, einzelne Befehle

rückgängig zu machen und zu wiederholen (*Undo/Redo*). In VIRTUALFLUIDS INTERACTIVE wird die Ausführung der Befehle und die Verwaltung der Historie von der Klasse `CommandProcessor` übernommen (vgl. Abbildung 4.13). Damit einzelne Befehlsobjekte auf diese Art verarbeitet werden kön-

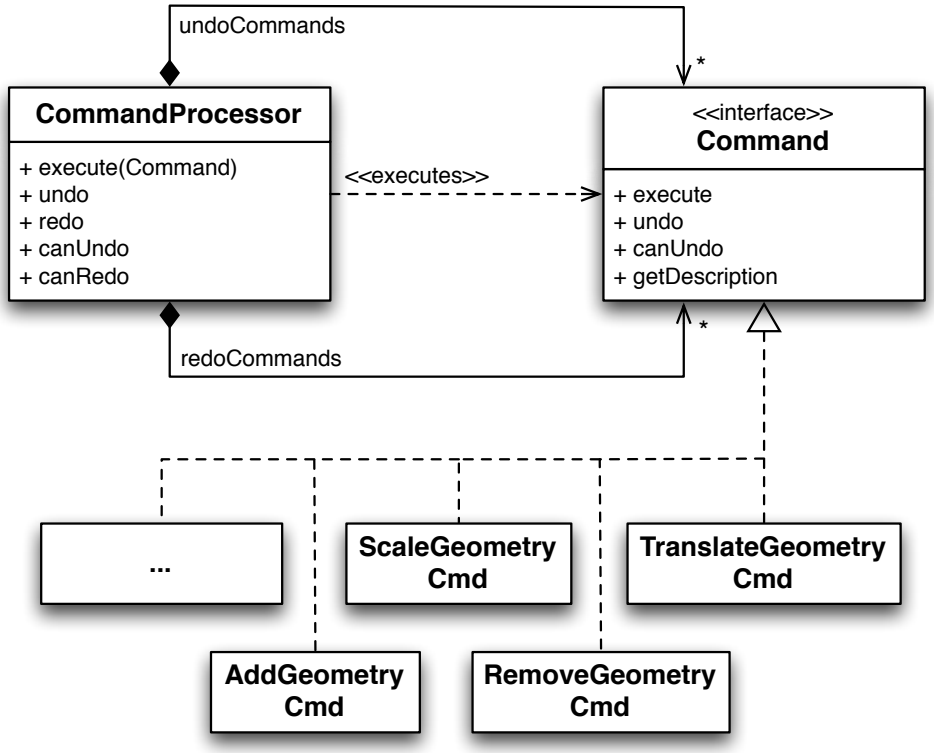


Abbildung 4.13: *Hierarchie von Befehlsklassen.*

nen, ist es erforderlich, dass sie die Schnittstelle `Command` umsetzen. Diese einheitliche Schnittstelle schreibt unter anderem vor, dass jedes Befehlsobjekt sowohl eine Methode zur Ausführung der Operationen anbietet als auch ein entsprechendes Gegenstück, das die angewendeten Operationen wieder rückgängig machen kann.

4.3.6 Persistenz

VIRTUALFLUIDS INTERACTIVE besitzt die Fähigkeit, Projekthierarchien dauerhaft im Dateisystem zu speichern. Zu diesem Zweck bietet die Datenschicht eine Komponente an, mit deren Hilfe Objekte unter Einsatz der Boost-Bibliothek serialisiert und persistiert werden können. Zusätzlich implementieren alle Elemente, die dauerhaft gespeichert werden können, einen Mechanismus, der es erlaubt, relevante Teile des internen Zustandes zu extrahieren. Dabei findet das Memento-Muster Anwendung, das es ermöglicht, den internen Zustand eines Objekts von dem eigentlichen Objekt zu trennen, ohne dabei die Datenkapselung aufzuheben (vgl. Abbildung 4.14). Der interne Zustand wird bei diesem Vorgehen von einem sogenannten Memento-Objekt repräsentiert, das eine Serialisierung und Deserialisierung der Zustandsbeschreibung ermöglicht. Zu diesem Zweck offerieren alle Memento-Objekte eine Methode `serialize`, die für eine Serialisierung mit Hilfe der Boost-Bibliothek erforderlich ist. Um das Speichern und Laden einer Projekthierarchie zu realisieren, bieten alle be-

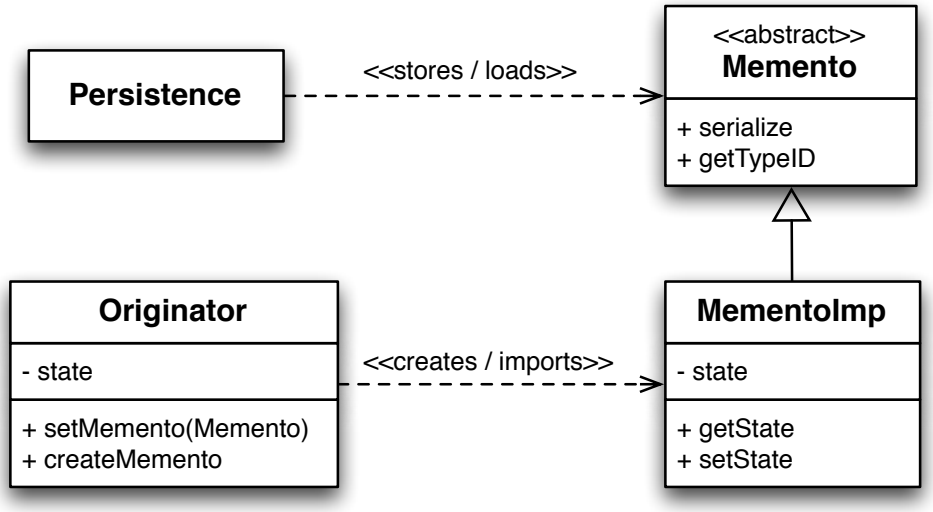


Abbildung 4.14: *Persistenzmechanismus für Projekthierarchien auf Basis des Memento-Musters.*

teiligten Komponenten die Möglichkeit, ihren internen Zustand in Form eines Memento-Objekts sowohl zu exportieren als auch zu importieren.

Durch Verwendung des Memento-Musters wird dabei erreicht, dass die Domänenobjekte dabei unabhängig von dem verwendeten Persistenzverfahren sind (SRP) und dieses nach Belieben ausgetauscht werden kann. Zusätzlich kann das Memento-Muster auch im Kontext der Befehlshistorie eingesetzt werden, um den internen Zustand eines Hierarchieelements festzuhalten und gegebenenfalls in diesen Zustand zurückversetzen zu können.

5 GPU-basierte numerische Strömungssimulation

Im Fokus des folgenden Kapitels steht die Lattice-Boltzmann-Methode als numerisches Verfahren zur Simulation von Strömungen und seine effiziente Implementierung zum Einsatz auf Grafikkarten. Dazu erfolgt zunächst eine Erläuterung der Grundlagen des numerischen Verfahrens, sowie der architekturellen Besonderheiten von GPUs und ihrer Programmierung.

5.1 Lattice-Boltzmann-Verfahren

Seit langem ist bekannt, dass die Navier-Stokes-Gleichungen zur allgemeinen Beschreibung und Vorhersage raum-zeitlicher Strömungs- und Transportprozesse herangezogen werden können [61, 136]. Dabei handelt es sich um ein System makroskopischer Erhaltungssätze, beschrieben durch partielle Differentialgleichungen, für die im Allgemeinen keine analytische Lösung bekannt ist. Ausgehend von den kontinuierlichen Gleichungen wählen klassische Ansätze zur näherungsweisen Lösung einen numerischen Diskretisierungsansatz in Raum und Zeit mehrheitlich mit Hilfe von Finite-Volumen-Verfahren [88]. Ein solcher Ansatz zur Lösung der makroskopischen Erhaltungssätze wird auch als *Top-Down*-Ansatz bezeichnet. Demgegenüber verfolgen Gittergas-Verfahren [148, 308] einen sogenannten *Bottom-Up*-Ansatz, dessen elementare Idee darin besteht, ein Fluid als Ensemble miteinander interagierender Partikel zu betrachten und das Verhalten eines Vielteilchensystems mit Hilfe künstlicher Mikrowelten aus »Partikeln« zu simulieren, die auf einem Berechnungsgitter angeordnet sind. Bei Betrachtung eines hinreichend großen Ensembles ist es dabei möglich, die hydrodynamischen Eigenschaften auf der makroskopischen Skala zu erhalten. Ein entscheidender Nachteil der Gittergas-Methode besteht jedoch darin, bei relevanten Problemgrößen einen enormen Speicherplatzbedarf zu erfordern und mit einem statistischen Rauschen behaftet zu sein. Die von McNamara und Zanetti [190] eingeführte und von anderen [120, 121] erweiterte Lattice-Boltzmann-Methode (LBM) verfolgt daher einen Ansatz auf der Meso-Skala, bei dem die Dynamik des Teilchenensembles nicht auf Grundlage einzelner Partikelbewegungen, sondern vielmehr anhand statistischer Aufenthaltswahrscheinlichkeiten der Teilchen beschrieben wird. Gegenüber der Gittergas-Methode ist das Verfahren wesentlich ressourcenschonender, so dass es in der Wissenschaft sehr viel Zuspruch findet [19, 56, 95, 236, 267].

5.1.1 Grundlagen

He, Luo und andere [119, 263] demonstrieren, dass die Lattice-Boltzmann-Methode als eine spezielle Diskretisierungsform der von Ludwig Boltzmann im Jahr 1872 aufgestellten Boltzmann-Gleichung (vgl. Gleichung 5.1) verstanden werden kann.

$$\frac{\partial f}{\partial t} + \vec{\xi} \cdot \frac{\partial f}{\partial \vec{x}} + \vec{F} \cdot \frac{\partial f}{\partial \vec{\xi}} = \Omega(f, f') \tag{5.1}$$

Die Boltzmann-Gleichung beschreibt die raum-zeitliche Entwicklung der Teilchen auf der mesoskopischen Skala anhand der Teilchenaufenthaltswahrscheinlichkeiten f. Dabei bestimmt $f = f(t, \vec{\xi}, \vec{x})$ die Wahrscheinlichkeit, ein Partikel mit der mikroskopischen Geschwindigkeit $\vec{\xi}$ am Ort $\vec{x}$ zum Zeitpunkt t anzutreffen. Die Interaktion der Teilchen wird durch den sogenannten Kollisionsoperator Ω abgebildet. Durch diesen wird die Boltzmann-Gleichung zu einer komplizierten Integro-Differentialgleichung, da die Aufenthaltswahrscheinlichkeiten der Teilchen nicht nur vom Ort, sondern auch von ihrer Geschwindigkeit abhängen. Ein direkter Lösungsansatz wäre weit aufwändiger als das Lösen der Navier-Stokes-Gleichungen selbst. Deswegen müssen für eine numerische Lösung weitere Vereinfachungen vorgenommen werden. Bhatnagar, Gross und Krook (BGK) [22, 227] wählen daher beispielsweise einen Ansatz, der die Details des Kollisionsprozesses vernachlässigt und von einem Zustand nahe dem Gleichgewicht ausgeht. Der Kollisionsoperator vereinfacht sich auf diese Weise zu:

$$\Omega = -\frac{1}{\tau}\left(f - f^{eq}\right) \tag{5.2}$$

Ausgehend von einem uniformen Gaszustand lassen sich die Aufenthaltswahrscheinlichkeiten der Teilchen mit Hilfe der Maxwellverteilungen beschreiben. Mit Hilfe des Kollisionsoperators werden diese Partikelverteilungen zum Gleichgewichtszustand, dem lokalen Maxwell'schen Gleichgewicht f^{eq}, relaxiert. Dabei bestimmt die Konstante τ die Relaxationszeit mit der die Annäherung an den Gleichgewichtszustand erfolgt. Zur weiteren Vereinfachung erfolgt eine Diskretisierung des Phasenraumes der Boltzmann-Gleichung hinsichtlich einer endlichen Menge diskreter mikroskopischer Geschwindigkeiten $\vec{e}_i$, wodurch man die diskrete Boltzmann-Gleichung erhält:

$$\frac{\partial f_i}{\partial t} + \vec{e}_i \frac{\partial f_i}{\partial \vec{x}} = -\frac{1}{\tau}\left(f_i - f_i^{eq}\right) \qquad i = 0, \ldots, (N-1) \tag{5.3}$$

Auf diese Weise wird die Verteilungsfunktion $f = f(t, \vec{\xi}, \vec{x})$ an N ausgewählten Kollokationspunkten durch die Werte $f(t, \vec{e}_i, \vec{x}) = f_i(t, \vec{x})$ dargestellt. Die Anordnung der Kollokationspunkte ergibt sich aus dem Satz von diskreten Geschwindigkeiten $\vec{e}_i$, durch die eine Einheitszelle eines kartesischen Gitters aufgespannt wird. Im zweidimensionalen Raum wird üblicherweise das sogenannte D2Q9-Modell mit neun diskreten Geschwindigkeiten verwendet (vgl. Abbildung 5.1).

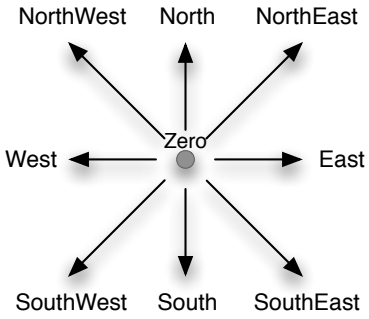
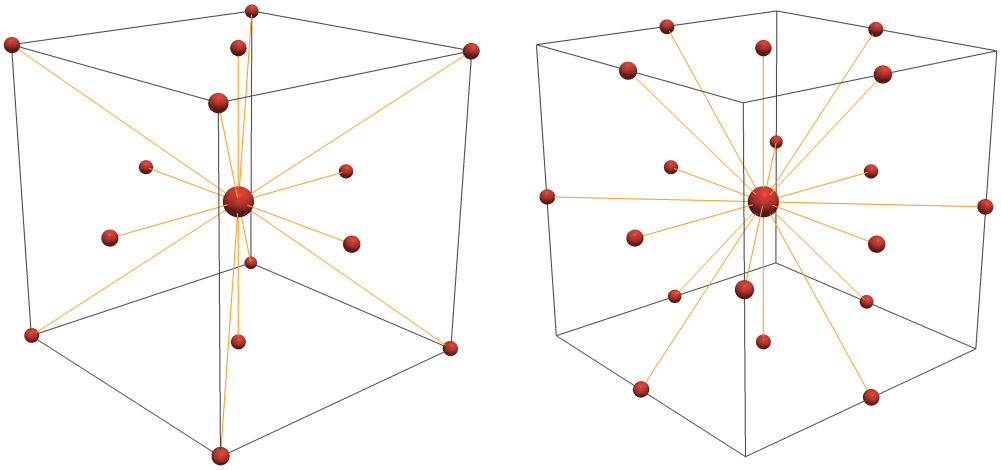


Abbildung 5.1: *D2Q9-Modell für LBM im zweidimensionalen Phasenraum.*

Die DdQq-Notation folgt dabei der von Qian [227] vorgeschlagenen Schreibweise, wobei d die Anzahl der Raumdimensionen und q die Anzahl an Kollokationspunkten angibt. Für Berechnungen

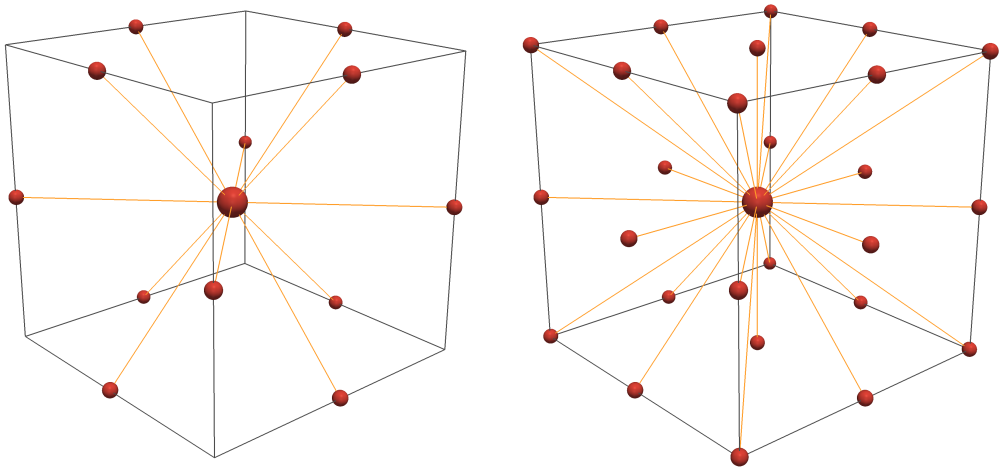
im dreidimensionalen Raum wird oftmals ein D3Q15- oder D3Q19-Modell eingesetzt (vgl. Abbildung 5.2). Gelegentlich wird das D3Q13-Modell (vgl. Abbildung 5.3) mit einem reduzierten Satz an



(a) *D3Q15-Modell* ***(b)*** *D3Q19-Modell*

Abbildung 5.2: *Diskrete Geschwindigkeitsvektoren häufig verwendeter LBM-Modelle im dreidimensionalen Raum.*

Geschwindigkeitsvektoren verwendet, das hinsichtlich der Implementierung einen Mehraufwand erfordert, jedoch ein hervorragendes Verhältnis von Genauigkeit und Ausführungsgeschwindigkeit aufweist [72, 281]. Ist eine hohe Stabilität und Genauigkeit gefordert, so kommt auch das D3Q27-Modell zum Einsatz.



(a) *D3Q13-Modell* ***(b)*** *D3Q27-Modell*

Abbildung 5.3: *Weniger häufig verwendete LBM-Modelle für hohe Ausführungsgeschwindigkeit bzw. Stabilität.*

In dieser Arbeit werden vor allem das D3Q13- und das D3Q19-Modell eingesetzt. Exemplarisch sind für das D3Q19-Modell die 19 Geschwindigkeitsvektoren für $i = 0, \ldots, 18$ aufgeführt:

$$\{\vec{e}_i\} = c \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \end{pmatrix}$$

Die Geschwindigkeitskonstante c ist eine frei wählbare Größe zur Skalierung der Geschwindigkeitsvektoren $\vec{e}_i$. Die Konstante c legt zugleich die Größe der Schallgeschwindigkeit c_s fest:

$$c_s = \frac{1}{\sqrt{3}} c \tag{5.4}$$

Die diskreten Boltzmann-Gleichungen werden nun anschließend mit einem finite Differenzenansatz raum-zeitlich diskretisiert, wobei der Gitterabstand dabei zu $\Delta x = c\Delta t$ gewählt wird, so dass man die Lattice-Boltzmann-Gleichung erhält:

$$f_i(t + \Delta t, \vec{x} + \vec{e}_i \Delta t) = f_i(t, \vec{x}) - \frac{\Delta t}{\tau} \left(f_i(t, \vec{x}) - f_i^{eq}(t, \vec{x}) \right) \tag{5.5}$$

Für äquidistante Gitter kann man vereinfachend $c = \Delta x = \Delta t = 1$ wählen und erhält folgendes Schema:

$$f_i(t + 1, \vec{x} + \vec{e}_i) = f_i(t, \vec{x}) - \frac{1}{\tau} \left(f_i(t, \vec{x}) - f_i^{eq}(t, \vec{x}) \right) \tag{5.6}$$

Die Gleichgewichtsverteilungen f_i^{eq} für inkompressible Strömungen werden mit Hilfe der Chapman-Enskog-Analyse [54] so bestimmt, dass die Physik der Navier-Stokes-Gleichungen beschrieben wird [119, 227]:

$$f_i^{eq}(t, \vec{x}) = w_i \left(\rho + \rho_0 \left(3 \frac{\vec{e}_i \cdot \vec{u}}{c^2} + \frac{9}{2} \frac{(\vec{e}_i \cdot \vec{u})^2}{c^4} - \frac{3}{2} \frac{u^2}{c^2} \right) \right), \tag{5.7}$$

Für das D3Q19-Modell erhält man:

$$w_i = \begin{cases} 1/3, & i = 0 \\ 1/18 & i = E, W, N, S, T, B \\ 1/36, & i = NE, SW, SE, NW, TE, BW, BE, TW, TN, BS, BN, TS \end{cases} \tag{5.8}$$

Die Indizes i ergeben sich dabei aus den Richtungen des D3Q19-Modells in Anlehnung an die Himmelsrichtungen (vgl. Tabelle 5.1).

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$d3q19$	Z	E	W	N	S	T	B	NE	SW	SE	NW	TE	BW	BE	TW	TN	BS	BN	TS

Tabelle 5.1: *Deklaration der Richtungen (E=East, W=West, N=North, S=South, T=Top, B=Bottom, Z=Zero)*

Von besonderem Interesse sind die makroskopischen Eigenschaften des Strömungsfeldes wie Masse und Geschwindigkeit. Diese Größen können aus den Momenten der Verteilungsfunktionen f näherungsweise durch Integration (und im Fall diskreter $\vec{e}_i$ durch Summation) bestimmt werden. Für Masse und Geschwindigkeit gilt:

$$\rho = \sum_i f_i(t,\vec{x}) \qquad \vec{u} = \frac{1}{\rho}\sum_i \vec{e}_i\, f_i(t,\vec{x}) \tag{5.9}$$

Für die kinematische Viskosität ν und die Relaxationszeit τ ergibt sich folgender Zusammenhang [277]:

$$\nu = \frac{2\tau - 1}{6} \tag{5.10}$$

Damit ergibt sich ein numerisches Schema, das in Hinblick auf die Implementierung anschaulich in zwei wesentliche Teilschritte untergliedert werden kann. Zur Lösung von Gleichung 5.6 wird in jedem Zeitschritt sowohl eine sogenannten Kollision als auch eine anschließende Propagation durchgeführt:

- In dem lokalen Kollisionsschritt wird am Ort $\vec{x}$ für den folgenden Zeitschritt $t+1$ der neue Verteilungssatz ermittelt:

$$f_i(t+1,\vec{x}) = f_i(t,\vec{x}) + \Omega_i(t,\vec{x}) \tag{5.11}$$

- Anschließend werden die neuen Verteilungen im Propagations- oder auch Advektionsschritt zu den unmittelbaren Nachbarknoten propagiert:

$$f_i(t+1,\vec{x}+\vec{e}_i) = f_i(t+1,\vec{x}) \tag{5.12}$$

Der Ablauf von Kollision und Propagation ist in Abbildung 5.4 für den zweidimensionalen Anwendungsfall grafisch veranschaulicht.

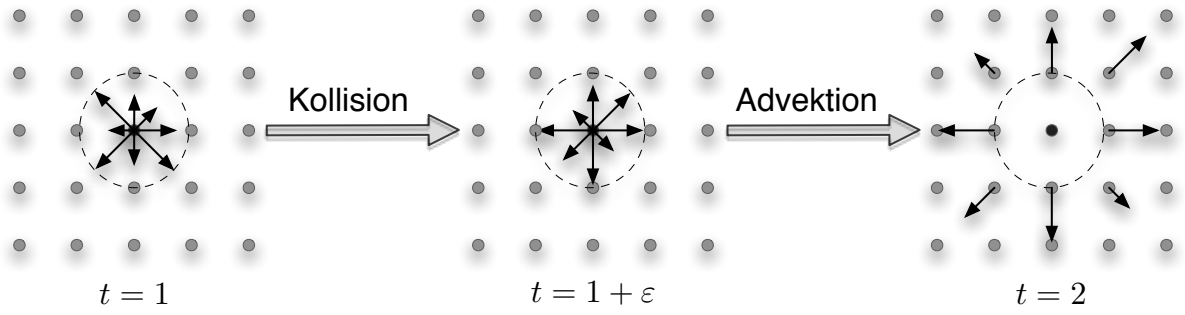


Abbildung 5.4: *Darstellung von Kollisions- und Propagationsschritt im D2Q9-Modell.*

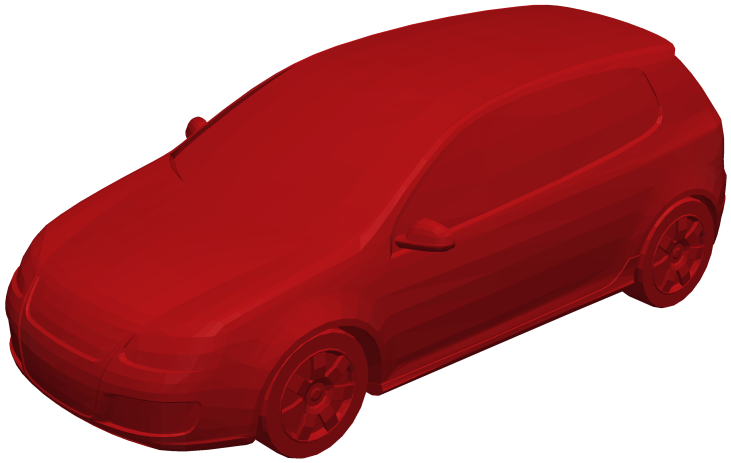
5.1.2 Randbedingungen

Für eine praktische Anwendung der Lattice-Boltzmann-Methode ist es letztlich notwendig, Randbedingungen für die Simulation zu definieren. Eine ausführliche Beschreibung von Randbedingungen mit Implementierungshinweisen kann beispielsweise [4, 94, 149] entnommen werden. Zu den

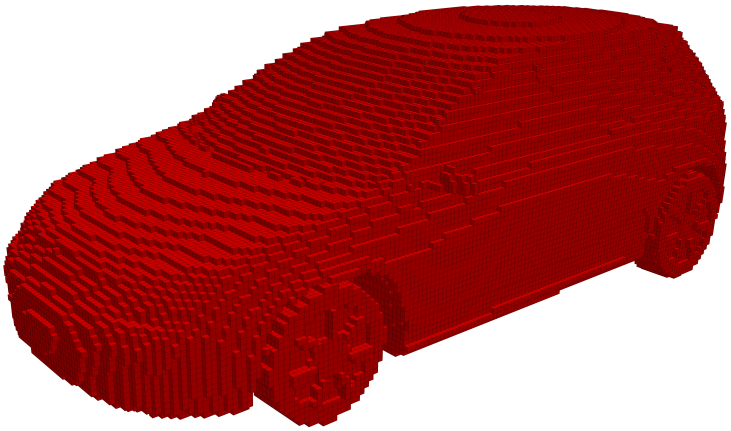
wichtigsten Randbedingungen zählen dabei Geschwindigkeitsrandbedingungen und Druckrandbedingungen. Die Abbildung von ortsfesten Strukturen geschieht über Haft-Randbedingungen, bei denen alle Geschwindigkeitskomponenten tangential zur Oberfläche zu Null gesetzt werden. Die einfachste Form von Haftrandbedingung ist die sogenannte *Bounce-Back*-Randbedingung. Dabei wird von der Annahme ausgegangen, dass die Struktur mit einer Gitterlinie zusammenfällt, so dass die Genauigkeit des Ansatzes auf die erste Ordnung beschränkt ist. Mit Hilfe geeigneter Interpolationsverfahren ist es jedoch möglich, eine höhere Genauigkeit zu erzielen [105]. Die räumliche Zuordnung von Randbedingungen auf Gitterknoten erfolgt im Allgemeinen in Anlehnung an den von Harlow und Welch vorgestellten *Marker-and-Cell*-Ansatz [116]. Das Verfahren sieht vor, jeden Gitterknoten mit einer Markierung zu versehen, mit deren Hilfe jeder Knoten entweder dem Fluid oder einer Randbedingung zugeordnet werden kann. Dafür ist eine Abbildung der geometrischen Strukturen der Ausgangsbeschreibung auf die Markerbeschreibung in Form einer Diskretisierung erforderlich. Dieser Vorgang wird allgemein auch als Voxelisierung bezeichnet. Analog zu zweidimensionalen Pixeln bezeichnet ein Voxel (Volumenelement) eine würfelförmige Zelle des zugrundeliegenden kartesischen Berechnungsgitters. Auf Grund der uniformen Struktur des Gitters kann dieser Vorgang im Gegensatz zu unstrukturierten Gittern in der Regel voll automatisiert ablaufen. Abbildung 5.5 zeigt am Beispiel eines Fahrzeuges die Ausgangsgeometrie sowie die diskretisierte Struktur in Form von Voxeln.

5.1.3 Alternative Kollisionsmodelle und Turbulenz

Typischerweise ist das Lattice-Boltzmann-Verfahren auf Strömungsphänomene mit kleiner Mach- und Knudsenzahl beschränkt [149]. Des Weiteren grenzt die verbreitete BGK-Approximation den möglichen Anwendungsbereich zusätzlich durch eine geringe minimale numerische Stabilität ein. Das von d'Humières et al. [73] vorgestellte Momentenmodell erlaubt demgegenüber eine deutlich geringere kinematische Viskosität und ermöglicht somit einen Einsatz bei größeren Gitter-Reynoldszahlen. Das Verfahren geht jedoch einher mit einem numerischen Mehraufwand, da die Verteilungen für die Kollision zunächst in den physikalisch äquivalenten Momentenraum transformiert werden. Anschließend werden anstelle der ursprünglichen Verteilungen die jeweiligen Momente relaxiert und in den Phasenraum zurücktransformiert. Im Gegensatz zum BGK-Verfahren, das für jedes Moment den selben Relaxationsparameter ansetzt, können die Parameter im Momentenmodell individuell gewählt werden. Das Verfahren wird daher auch als *Multiple-Relaxation-Time*-Modell (MRT) bezeichnet. Eine weitere Steigerung der Stabilität, kann durch Verwendung des von Geier [99, 100] vorgestellten Kaskadierten LB-Modells (CLB) (*Cascaded Lattice Boltzmann*) erreicht werden. Ebenso wie im Momentenmodell findet die Relaxation bei diesem Verfahren im Momentenraum statt. Anstelle die Momente jedoch wie im MRT-Modell üblich im Ruhesystem zu entwickeln, verwendet das CLB-Verfahren zentrale Momente, die im lokalen Schwerpunktbezugssystem gebildet werden. Die meisten praxisrelevanten Strömungen sind jedoch stark turbulent und weisen somit sehr hohe Reynoldszahlen auf. Um die Stabilität dennoch zu gewährleisten, ist es beispielsweise möglich, das *Large-Eddy*-Simulationsmodell (LES) zur Modellierung der Turbulenz zu integrieren [125]. Aktuelle Arbeiten beschäftigen sich zudem mit hierarchischen Ansätzen zur Gitterverfeinerung und Adaptivität [66, 101, 104, 243, 282].



(a) *Facettiertes Dreiecksgitter (STL) als Ausgangsbeschreibung der Geometrie.*



(b) *Voxelisierte Darstellung als Beschreibung der Randbedingung in der Simulation.*

Abbildung 5.5: *Voxelisierung eines Fahrzeuges zur Repräsentation als Haft-Randbedingung.*

5.1.4 Leistungsanforderungen

Ingenieurtechnisch relevante Strömungen von Luft und Wasser sind häufig durch hohe Reynoldszahlen $\mathrm{Re} = \frac{u_0 L_0}{\nu_0} = \mathcal{O}(10^5 - 10^7)$ gekennzeichnet. Im Prinzip können auch diese turbulenten Regime adäquat von den Navier-Stokes-Gleichungen beschrieben werden. Dies impliziert jedoch, dass das Geschwindigkeitsfeld eines turbulenten Strömungsproblems alle dynamischen Strukturen wie beispielsweise Wirbel beinhaltet. Die erforderlichen minimalen Zeit-, Längen- und Geschwindigkeitsskalen können nach Kolmogorov [147] unter Verwendung der kinematischen Viskosität ν_0 $[\frac{m^2}{s}]$ und der Dissipationsrate ϵ $[\frac{m^2}{s^3}]$ wie folgt abgeschätzt werden:

$$L_s = \left(\frac{\nu_0^3}{\epsilon}\right)^{\frac{1}{4}} \quad , T_s = \left(\frac{\nu_0}{\epsilon}\right)^{\frac{1}{2}} \quad , U_s = \left(\frac{\nu_0}{\epsilon}\right)^{\frac{1}{2}}$$

Sollen alle Strukturen einer turbulenten Strömung durch eine sogenannte direkte numerische Simulation (DNS) aufgelöst werden, muss das Berechnungsgitter die größten und kleinsten raumzeitlichen Strukturen widerspiegeln. Hinsichtlich der Anzahl notwendiger Gitterknoten N_G und Zeitschritte N_T resultieren daraus folgende Anforderungen:

$$N_G \simeq \mathrm{Re}^{\frac{9}{4}} \tag{5.13}$$

$$T_G \simeq \mathrm{Re}^{\frac{3}{4}} \tag{5.14}$$

Für die Berechnung der Umströmung eines Gebäudes ergibt sich daraus schon bei moderaten Windgeschwindigkeiten und $\mathrm{Re} \simeq 10^7$ eine notwendige Anzahl von 10^{21} Knotenaktualisierungen. Selbst bei Einsatz moderner Supercomputer ist dies auf absehbare Zeit eine unlösbare Aufgabe, weshalb alternativ Turbulenzmodelle zum Einsatz kommen. Diese berücksichtigen einen mehr oder minder großen Anteil der beteiligten raumzeitlichen Skalen in Form implizit modellierter Einflussgrößen auf die explizit diskretisierten Skalen. Verallgemeinert lässt sich sagen, dass sich der Simulationsaufwand invers zum Modellieraufwand verhält. Je mehr Skalen explizit berücksichtigt werden sollen, desto mehr Gitterknoten und Zeitschritte werden erforderlich.

Selbst leistungsfähige Computer von Lattice-Boltzmann-Simulationen bei zunehmender Problemgröße leicht an die Grenzen ihrer Leistungsfähigkeit getrieben werden können. Diese Aussage gilt gleichsam für die Rechenleistung wie auch für den Speicherplatzbedarf. Zur Verdeutlichung sei an dieser Stelle beispielhaft ein System mit 10^9 Gitterknoten betrachtet. Ausgehend von dem D3Q19-Modell mit jeweils 19 Freiheitsgraden pro Gitterknoten und der Verwendung der Momentenmethode werden dabei pro Zeitschritt an jedem Knoten abhängig von dem verwendeten Compiler ca. 175-200 Rechenoperationen durchgeführt [297]. In der Summe erfordert folglich jeder Zeitschritt die Ausführung von ca. $2 \cdot 10^{11}$ Rechenoperationen. Demgegenüber kann eine aktuelle Intel Core i7 CPU theoretisch ca. 10^{11} Rechenoperationen in der Sekunde ausführen. Ausgehend von der vollen theoretischen Leistung des Prozessors würde die Berechnung eines Zeitschritts ca. 2 Sekunden benötigen und dabei ca. 140 GB Speicher belegen. Eine typischen LB-Geschwindigkeit von $0,1\ [\frac{\Delta x}{\Delta t}]$ zugrunde gelegt, würde die Umlaufzeit bezogen auf das gesamte Gitter damit mehr als 5,5 Stunden betragen. Auf Grund der mit zunehmender Problemgröße rasant ansteigenden Hardwareanforderungen (vgl. Tabelle 5.2) stellen dreidimensionale numerische Strömungssimulationen daher einen klassischen Anwendungsfall des Hochleistungsrechnens dar.

Gitterknoten	Rechenoperationen pro Zeitschritt	Speicherbedarf
10^6	$2 \cdot 10^8$	145 MB
10^9	$2 \cdot 10^{11}$	142 GB
10^{12}	$2 \cdot 10^{14}$	138 TB

Tabelle 5.2: *Hardwareanforderungen dreidimensionaler LB-Simulationen bei unterschiedlichen Gitterauflösungen.*

5.2 GPU-basiertes Hochleistungsrechnen

Sogenannte Hochleistungsrechner oder auch Supercomputer zeichnen sich zum Zeitpunkt ihrer Installation durch eine überragende Rechenleistung und Speicherkapazität aus. Der Begriff »Supercomputer« wurde erstmals 1920 von der US-amerikanischen Zeitung »New York World« zur Beschreibung einer Rechenmaschine verwendet, die von der Firma IBM speziell für die Universität Columbia gefertigt worden war [13]. In den Jahren zwischen 1975 und 1995 wurde das Hochleistungsrechnen von Firmen wie Cray, CDC, NEC, Fujitsu oder Thinking Machines dominiert. Die von ihnen entwickelten Systeme waren speziell auf den HPC-Markt zugeschnitten und den allgemein als *Personal Computer* (PC) bekannten konventionellen Systemen hinsichtlich ihrer Rechenleistung weit überlegen. Jedoch waren derartige Systeme auch mit enormen finanziellen Investitionen verbunden. Heute wird der Sektor des Hochleistungsrechnens hingegen von kosteneffektiven Systemen dominiert, deren Komponenten weniger für den Einsatz im wissenschaftlichen Rechnen, als vielmehr für den Massenmarkt entwickelt werden [113]. Dennoch ist das konventionelle Hochleistungsrechnen nach wie vor mit erheblichen Kosten verbunden, wodurch es allgemein größeren Unternehmen und akademischen Einrichtungen vorbehalten bleibt. Die hohen Leistungen der Supercomputer basieren im Allgemeinen auf einer Parallelisierung der Berechnungen, bei der mehrere, häufig verteilte Ressourcen simultan zur Lösung einer Aufgabe eingesetzt werden. Die notwendigerweise parallele Anwendungsentwicklung für derartige Systeme ist jedoch erheblich komplexer als die Entwicklung sequentieller Applikationen und erschwert zudem die Fehlersuche und Leistungsoptimierung. Die Ursache dafür ist das geringe hardwarenahe Abstraktionsniveau, auf dem parallele Anwendungen entwickelt werden. Die gebräuchlichen Verfahren sind häufig sehr fehleranfällig und bieten wenig Raum für Wiederverwendung, so dass die Produktivität tendenziell eher eingeschränkt ist [158]. Auf Grund seiner geringen Verbreitung und der erforderlichen parallelen und verteilten Programmierparadigmen galt das Hochleistungsrechnen daher lange Zeit als »exotisch«. Seit Aufkommen der sogenannten *Multicore*- oder auch *Concurrency*-Revolution [269] sind die parallelen Programmierkonzepte jedoch auch in der allgemeinen Anwendungsentwicklung allgegenwärtig.

5.2.1 Datenparallelität

Wie durch das Mooresche Gesetz [194] beschrieben, verdoppelt sich die Leistung der Hauptprozessorgenerationen in etwa alle 18 bis 24 Monate. Noch bis vor wenigen Jahren wurde dieser Leistungszuwachs durch eine Verdopplung der Transistoren eines einzelnen Chips erzielt, wobei damit auch eine Erhöhung der Taktrate einherging. Diese kontinuierliche Entwicklung ist jedoch insofern limitiert, als höhere Taktraten einen höheren Energieverbrauch bedingen und sich gezeigt hat, dass eine effiziente Kühlung lediglich bis zu einer Taktrate von etwa 3 GHz möglich ist. Aus diesem Grund gingen Hersteller von Hauptprozessoren (CPUs) (*Central Processing Unit*) wie Intel und AMD im Jahr 2003 dazu über, sogenannte Mehrkernprozessoren zu entwickeln. Das Gesetz von Moore besitzt auch für diese Prozessorengeneration nach wie vor Gültigkeit, jedoch führt jede Verdopplung der Transistoren zu einer gleichzeitigen Verdopplung der Prozessorkerne. Heute verfügbare Hauptprozessoren haben in der Regel vier oder mehr Kerne. Im Fall eines Intel Core i7-980X Prozessors sind beispielsweise sechs Kerne vorhanden, die zusammen eine theoretische Leistung von bis zu 107,55 GFLOPS ($107,55 \cdot 10^9$ Fließkommaoperationen pro Sekunde) erzielen. Um die Rechenleistung aller Kerne aus-

zuschöpfen, müssen sich Entwickler jedoch notwendigerweise paralleler Programmierkonzepte bedienen.

Eines der ältesten und etabliertesten parallelen Konzepte ist die datenparallele Programmierung. Bei diesem Paradigma wird eine Operation oder auch eine Sequenz von Operationen simultan auf jedes Element eines Datensatzes angewandt [26]. Die Granularität dieser Operationen kann dabei von einer einfachen Addition zweier Zahlen über eine Matrixmultiplikation bis zu komplexen Visualisierungsalgorithmen variieren. Eine besondere Attraktivität der datenparallelen Programmierung besteht darin, dass die Parallelität zu einem überwiegenden Maße vor dem Entwickler verborgen bleibt [158]. Mehrere Berechnungsstränge werden isoliert ausgeführt und arbeiten dabei auf dedizierten Elementen eines Datensatzes. Die auszuführenden Operationen werden im Quelltext jedoch sequentiell spezifiziert. Zusätzlich nimmt die Laufzeitumgebung dem Entwickler die Aufgabe ab, sich um Details des Datenaustauschs oder der Verwaltung der Berechnungen Gedanken machen zu müssen. Grafikprozessoren (GPUs) bieten im Gegensatz zu CPUs eine hardwareseitige Unterstützung für datenparallele Algorithmen, da sie eine sogenannte SIMD-Architektur (*Single Instruction, Multiple Data* [89]) implementieren. Aktuelle Generationen erlauben zudem eine flexible Programmierung und damit die Ausführung effizienter numerischer Berechnungen. In diesem Zusammenhang wird häufig auch von GPGPU-Programmierung (*General Purpose Computation on Graphics Processing Unit* [143]) oder allgemeiner von GPU-Computing gesprochen.

Angetrieben von der Spieleindustrie erfahren Grafikprozessoren seit Mitte der 1990er Jahre eine stetige Leistungssteigerung. Da die SIMD-Architektur eine effektive Nutzung zusätzlicher Transistoren durch einfache Erhöhung der Pipelineanzahl ermöglicht, verdoppelt sich die Leistung der GPUs im Gegensatz zu CPUs jedoch bereits alle sechs Monate. Für die Zukunft erwartet man, dass dieser Trend weiterhin Bestand haben wird [85]. Eine heute aktuelle NVIDIA Tesla C2070 Karte erzielt beispielsweise eine theoretische Leistung von $1,03 \cdot 10^{11}$ Fließkommaoperationen pro Sekunde (1,03 TFLOPS). Dies gilt jedoch nur für Berechnungen einfacher Genauigkeit. Bei doppelter Genauigkeit reduziert sich die Leistung auf 515 GFLOPS. Die im Rahmen dieser Arbeit vorwiegend eingesetzte Tesla C1060 erreicht bei einfacher Genauigkeit eine theoretische Leistung von 936 GFLOPS. Bei doppelter Genauigkeit liegt ihre maximale Leistung hingegen bei lediglich 78 GFLOPS und somit unterhalb derer einer leistungsfähigen CPU. Diese Tatsache unterstreicht, dass GPUs ursprünglich primär für den Einsatz einfach genauer Berechnungen vorgesehen sind.

In einem einzelnen System lässt sich die Leistung mehrerer Grafikprozessoren mit geringem Aufwand kombinieren. Auf diese Weise sind leistungsfähige Arbeitsplatzrechner realisierbar, die hinsichtlich der theoretischen Maximalleistung in Regionen des Hochleistungsrechnen vorstoßen. Das im Rahmen dieser Arbeit eingesetzte System ist beispielsweise mit vier Tesla-Karten ausgestattet und erreicht damit theoretisch mehr als vier TFLOPS. Derartige Karten werden von NVIDIA speziell für den Einsatz im wissenschaftlichen Rechnen angeboten und zeichnen sich insbesondere durch einen großen fehlertoleranten Speicher aus. Gleichzeitig markieren sie jedoch auch das obere Preissegment heute zur Verfügung stehender Grafikadapter. Grundsätzlich kann die selbe Rechenleistung auch von günstigeren »*Consumer*«-Produkten erzielt werden. Diese Karten besitzen jedoch Einschränkungen hinsichtlich der Größe und Qualität des Speichers sowie der Leistung bei doppelter Genauigkeit. Weitaus höherer finanzieller, administrativer und infrastruktureller Aufwand müsste jedoch investiert werden, würde man beabsichtigen, eine vergleichbare Leistung auf konventioneller CPU-Basis beispielsweise unter Einsatz eines Computer-Clusters zu erzielen. Allerdings kann die GPU-Technik

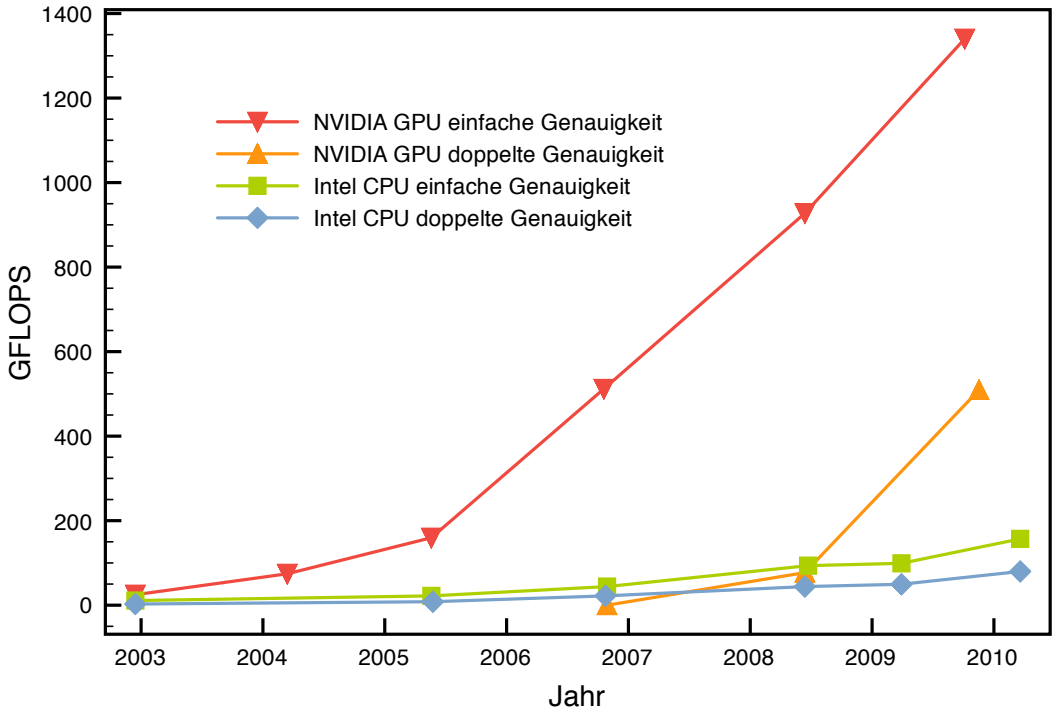


Abbildung 5.6: *Zunehmend divergierende Rechenleistungen von CPUs und GPUs. (Die Daten wurden [208] entnommen.)*

auch dazu verwendet werden, eben solche Cluster-Systeme mit zusätzlicher Rechenleistung auszustatten. Die auf diese Weise entstehenden hybriden Systeme können Leistungen im Petaflop-Bereich erzielen [69].

5.2.2 Architekturelle Unterschiede von GPU und CPU

Das zum Teil gravierende Ungleichgewicht der Leistungen von CPUs und GPUs begründet sich in den fundamental unterschiedlichen Entwurfsphilosophien beider Prozessortypen. Traditionell werden die meisten Anwendungen nach dem Referenzmodell von Neumanns [199] als sequentielle Programme entwickelt. Heute verfügbare konventionelle Mehrkernprozessoren sind zumeist superskalare CPUs, die speziell für die Ausführung derartiger Programme optimiert sind. Die einzelnen CPU-Kerne sind dabei sehr mächtig. Jeder einzelne implementiert einen vollen x86-Befehlssatz und unterstützt Techniken wie die *Out-of-Order Execution*[1] (OOE) und *Simultaneous Multithreading*[2] (SMT), um die Ausführungsgeschwindigkeit sequentieller Anwendungen zu maximieren. Die dafür notwendige anspruchsvolle Kontrolllogik nimmt jedoch einen erheblichen Anteil des Chips ein (vgl. Abbil-

[1] Die OOE ist ein Paradigma, das in Mikroprozessoren eingesetzt wird, um Befehlszyklen zu nutzen, die andernfalls auf Grund von Latenzen ungenutzt blieben. Zu diesem Zweck werden die Befehle in den Ausführungseinheiten des Prozessors außerhalb der ursprünglichen Reihenfolge ausgeführt [256].

[2] Der simultane Mehrfadenbetrieb (*Simultaneous Multithreading*) ist eine Technik, um mittels hardwareseitigem *Multithreading* die Gesamteffizienz superskalarer Prozessoren zu verbessern, indem mehreren simultan ausgeführten Programmfäden ermöglicht wird, die Ressourcen moderner Prozessoren besser auszunutzen [272, 286].

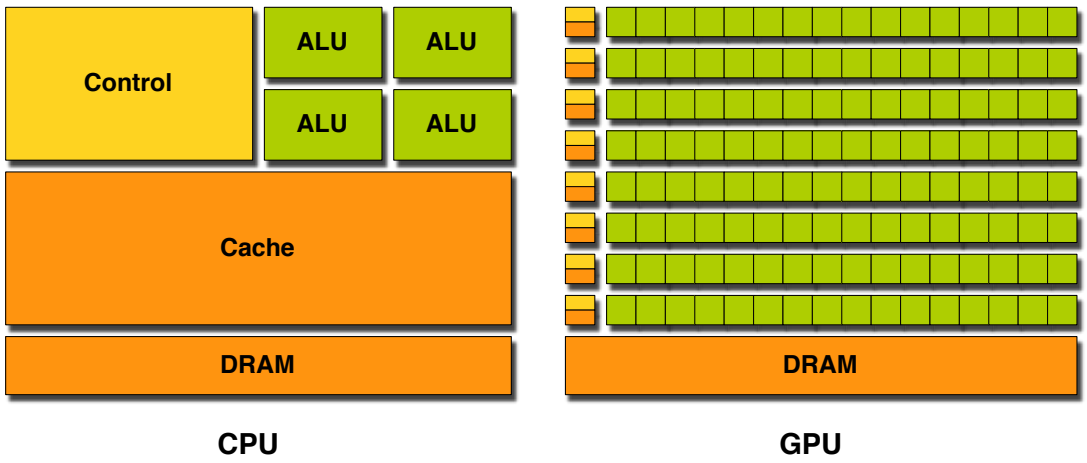


Abbildung 5.7: *Schematische Darstellung der Verteilung von Transistoren bei CPUs und GPUs basierend auf [208].*

dung 5.7). Noch mehr Bedeutung haben nur die großen Cache-Speicher, die vorgehalten werden, um Latenzen beim Zugriff auf Daten und Instruktionen zu reduzieren. Jedoch tragen zunächst weder die Kontrolllogik noch die Caches unmittelbar zu einer Steigerung der Rechenleistung bei.

Grafikprozessoren hingegen widmen die überwiegende Mehrzahl ihrer Transistoren den Recheneinheiten[3] (ALUs). Diese sogenannten Many-Core-Architekturen sind hinsichtlich der Ausführung paralleler Algorithmen und eines hohen Datendurchsatzes optimiert. Im Fall eines GT200-Prozessors, der unter andrem auf Tesla C1060 Karten eingesetzt wird, teilen sich jeweils acht der vorhandenen 240 Kerne eine Kontroll- und Befehlseinheit. Die acht Kerne, oder auch Streaming-Prozessoren (SP), formen auf diese Weise einen Streaming-Multiprozessor (SM), der dafür ausgelegt ist, eine immense Anzahl Ausführungsstränge, sogenannte *Threads*, simultan zu verarbeiten. So unterstützt ein GT200-Chip beispielsweise bis zu 1024 aktive *Threads* pro SM was mehr als 30.000 *Threads* pro Chip entspricht. Im Vergleich dazu können auf einem Intel-Prozessor abhängig vom Modell lediglich zwei bis vier *Threads* pro Kern simultan aktiv sein. Die zahlreichen *GPU-Threads* sind im Gegensatz zu *CPU-Threads* jedoch sehr leichtgewichtig. Zudem werden für jeden aktiven GPU-Thread separate Register allokiert, so dass ein Streaming-Multiprozessor einen unterbrechungsfreien Kontextwechsel durchführen kann. Diese Technik wird auch als *»Zero-Overhead Thread Scheduling«* bezeichnet und ist von entscheidender Bedeutung für die Fließkommaperformance [143]. Sie verleiht der GPU die Fähigkeit zum sogenannten *Latency Hiding*. Auf Grund der hohen Skalierbarkeit der Threadverwaltung und der großen Anzahl aktiver *Threads* wird ein Streaming-Multiprozessor somit in die Lage versetzt, aus Speicherzugriffen resultierende Latenzen zu überdecken. Zu diesem Zweck werden *Threads*, die auf angeforderte Daten warten, in den Hintergrund gestellt, während andere, zur Ausführung bereite *Threads*, reaktiviert werden. Dieses Verfahren arbeitet dann sehr effizient, wenn das Verhältnis von aktiven *Threads* zu Ausführungseinheiten (SPs) groß ist. Die Fähigkeit, Operationen mit hohen Latenzen tolerieren zu können, ist einer der Hauptgründe dafür, dass GPUs im Vergleich zu CPUs deutlich kleinere Caches und schlankere Kontrolllogik benötigen. Stattdessen widmen sie den bei weitem größten Anteil an Transistoren den Recheneinheiten, wodurch sich die höhere theoretische

[3] Als arithmetisch-logische Einheit (*Arithmetic Logic Unit*) oder auch Recheneinheit bezeichnet man einen elektronischen Schaltkreis, der arithmetische und logische Operationen ausführen kann [261].

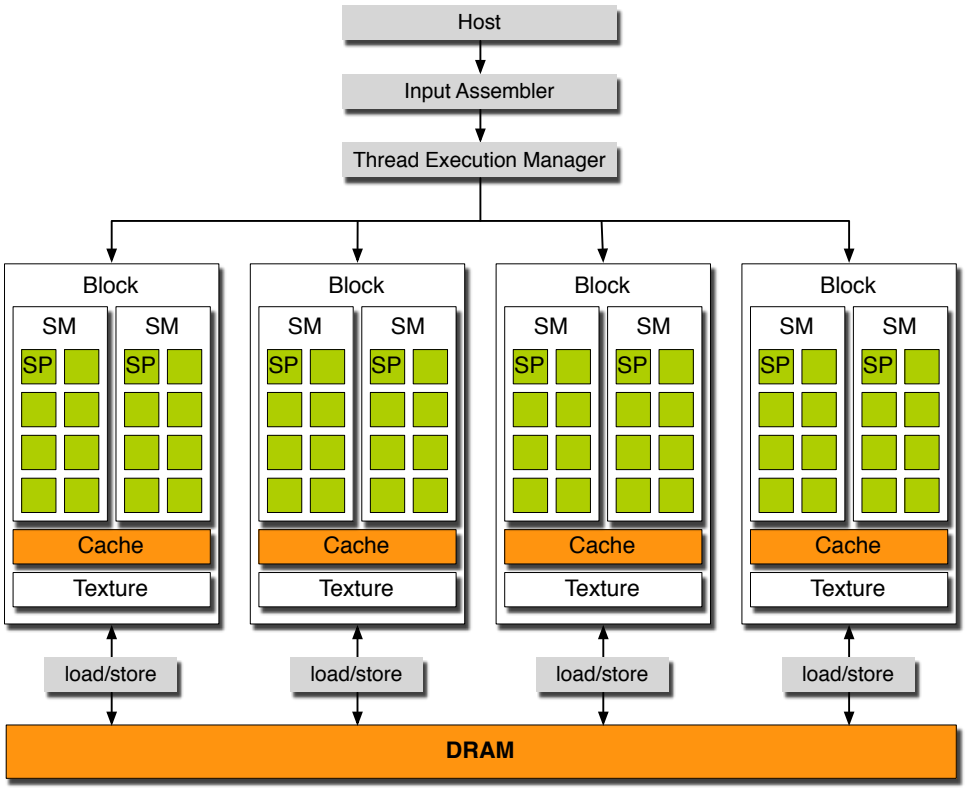


Abbildung 5.8: *Schematische Darstellung der Architektur einer CUDA-fähigen GPU basierend auf [143].*

Rechenleistung erklärt. Grafikprozessoren sind CPUs jedoch nicht nur hinsichtlich der Rechenleistung überlegen. Die Hersteller sind seit jeher bemüht, eine möglichst performante Anbindung der GPUs an den Grafikspeicher zu realisieren, um auch bei hohen Bildschirmauflösungen mit beeindruckenden Bildwiederholungsraten aufwarten zu können. Da die Latenz des Speicherzugriffs dabei ohne weiteres von der massiv parallelen Ausführung verdeckt werden kann, ist es möglich, die Speicheranbindung hinsichtlich der Bandbreite zu optimieren und Latenzen anders als bei CPUs zu vernachlässigen. Aktuelle Entwicklungen erreichen nicht selten eine theoretische Datentransferrate von mehr als 100 GB/s und liegen damit um ein Mehrfaches über der Leistung aktueller CPUs (vgl. Abbildung 5.9), von denen angenommen wird, dass sie in den nächsten drei Jahren nicht über 50 GB/s hinauswachsen werden [143]. Damit wird deutlich, dass GPUs speziell für die Ausführung massiv paralleler Berechnungen ausgelegt und in dieser Hinsicht konventionellen Hauptprozessoren im Allgemeinen überlegen sind. Sie bieten jedoch weit weniger algorithmische Flexibilität als CPUs und stehen diesen insbesondere bei der Ausführung sequentieller Anwendungen nach. Um die Vorteile beider Paradigmen kombinieren zu können, ist das von NVIDIA im Jahre 2007 vorgestellte CUDA-Programmiermodell [208] für eine heterogene CPU-GPU-Ausführung ausgelegt.

5.2.3 CUDA-Programmiermodell

Die CUDA-Entwicklungsumgebung ist mittlerweile für alle gängigen Plattformen - Windows, Linux und OS X - verfügbar. Mit ihrer Hilfe ist es Entwicklern möglich, die Leistung moderner GPUs in eigenen Applikationen zu nutzen. Dabei basiert das Programmiermodell im Wesentlichen auf der Hoch-

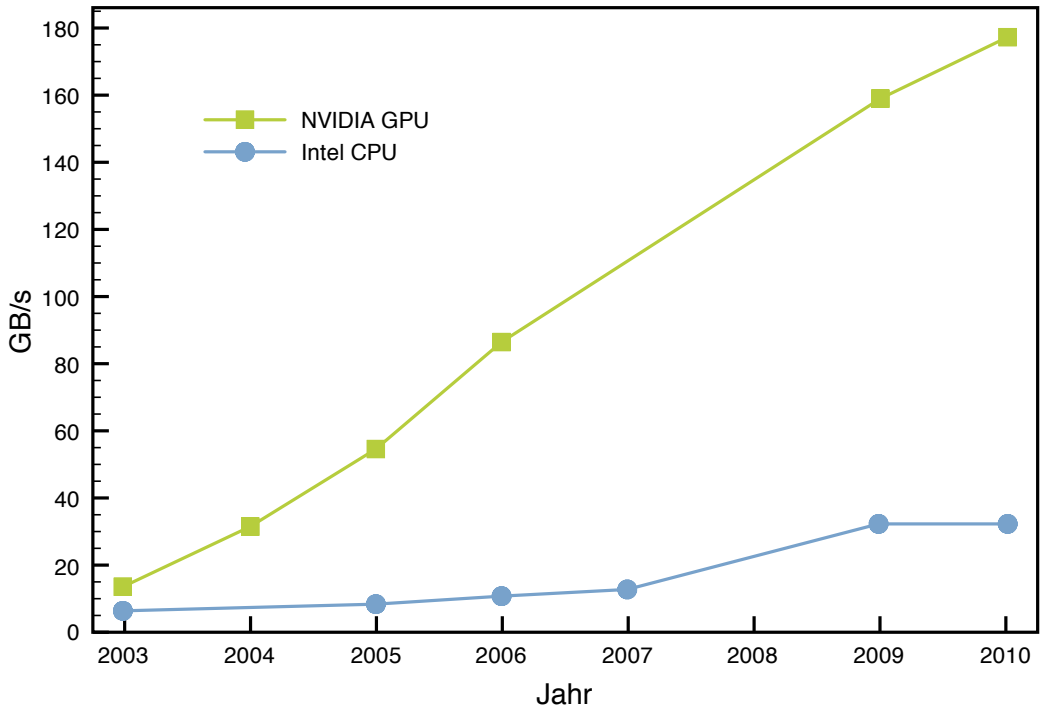


Abbildung 5.9: *Vergleich der theoretischen Speicherbandbreite von GPUs und CPUs aufgetragen über der Zeit. (Die Daten wurden [208] entnommen.)*

sprache C und erfordert lediglich eine flache Lernkurve von Entwicklern, die mit der Sprache vertraut sind. Die Schlüsselabstraktionen in Form einer Hierarchie von Threadgruppen, gemeinsamer Datenspeicher und eines Synchronisierungsmechanismus werden durch eine minimale Erweiterung der Sprache gebildet. Sie ermöglichen eine feingranulare Daten- und Threadparallelität, die eine sehr gute Skalierbarkeit auf einer diversifizierten Hardwarebasis ermöglicht. Das heterogene Programmiermodell erachtet die als *»Device«* bezeichneten GPUs dabei als Coprozessor für die CPU. Diese wird im Kontext von CUDA auch *»Host«* genannt. Für jedes der Geräte wird ein separater dynamischer Datenspeicher (DRAM) vorgehalten, der als *Device*-Speicher respektive *Host*-Speicher bezeichnet wird. Die CUDA-Programmierschnittstelle, auch bezeichnet als CUDA-API (*Application Programming Interface*), erlaubt ein effizientes Kopieren der Daten vom einen in den anderen Speicher mittels optimierter Routinen unter Verwendung des Speicherdirektzugriffs[4] (DMA). Gleichzeitig führt die CUDA-C-Programmiersprache einen neuen Funktionstypen ein, der ein datenparalleles Paradigma ermöglicht. Sogenannten *Kernel*-Funktionen werden bei einem Aufruf N mal von N CUDA-*Threads* parallel ausgeführt. Jedem *Thread* steht dabei ein exklusiver lokaler Speicher in Form von Registern zur Verfügung. Zusätzlich kann jeder Thread auf den gemeinsam verwendeten globalen Hauptspeicher zugreifen. Eine logische Zuordnung von Elementen auf dem Hauptspeicher und einzelnen *Threads* im Sinne des SIMD-Konzeptes wird dabei über einen eindeutigen Threadindex ermöglicht. Um eine

[4] Als *Direct Memory Access* bezeichnet man die Fähigkeit moderner Computer und Mikroprozessoren, bestimmten Hardwarekomponenten den direkten, von der CPU unabhängigen Schreib- und Lesezugriff auf den Hauptspeicher zu ermöglichen [261].

möglichst gute Skalierbarkeit zu erreichen, führt CUDA eine Parallelität auf zwei Ebenen ein. Einzelne *Threads* werden zunächst zu gleich großen ein-, zwei- oder dreidimensionalen Blöcken (*Blocks*) gruppiert. Auf diese Weise wird eine natürliche Abbildung auf Elemente der Anwendungsdomäne wie Vektoren, Matrizen oder Tensoren erreicht. Threadblöcke sind weiterhin in ein- oder zweidimensionalen Gittern (*Grids*) organisiert (vgl. Abbildung 5.10), deren Größe sich in der Regel aus den Dimensionen des Berechnungsgebiets ergibt. *Threads* innerhalb eines Blocks weisen die Besonderheit auf,

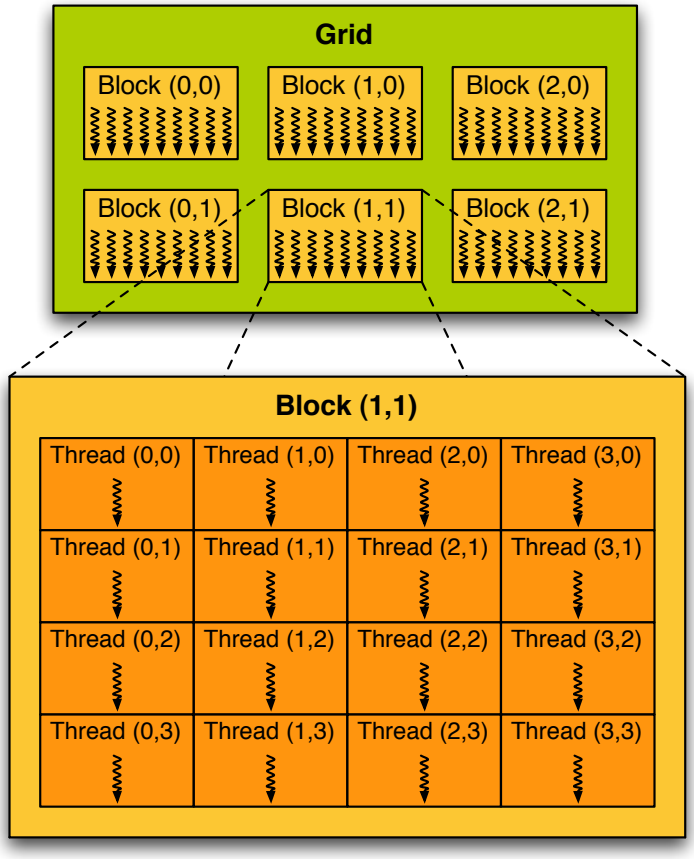


Abbildung 5.10: *Hierarchische Organisation von* CUDA-Threads *mit Hilfe von Blöcken und Gittern basierend auf [208].*

Daten über einen sehr schnellen gemeinsamen Speicher (*Shared Memory*) ähnlich eines L1-Caches, austauschen zu können. Diese Fähigkeit ergibt sich daraus, dass alle *Threads* innerhalb eines Blocks auf dem selben Streaming-Multiprozessor ausgeführt werden. Die Blöcke eines *Grids* werden nach Möglichkeit gleichmäßig über die Multiprozessoren verteilt, um eine hohe Performance zu erzielen.

5.3 CUDA-basierte LB-Simulation

Die Lattice-Boltzmann-Methode ist, wie in Abschnitt 5.1 beschrieben, ein von Natur aus explizites Verfahren, das auf einem Finite-Differenzen-Gitter arbeitet und im Allgemeinen lediglich eine Kommunikation mit den direkten Nachbarknoten erfordert. Auf Grund dieser Eigenschaften ist das Verfahren sehr gut für eine datenparallele Implementierung und dadurch insbesondere für eine Ausfüh-

rung auf GPUs geeignet. So finden sich zahlreiche Arbeiten, die sich bereits vor der Veröffentlichung von CUDA mit der Implementierung der LB-Methode auf GPU-Basis befassen, obwohl zu dieser Zeit noch eine grafikorientierte und hardwarenahe Form der Programmierung erforderlich war. Li und andere [161] stellen beispielsweise eine GPU-beschleunigte LBM-Implementierung vor, bei der die Verteilungen vollständig im 2D-Texturspeicher vorgehalten werden und die Boltzmann-Gleichungen in Form von Bildspeicher- und Rasterisierungsoperationen abgebildet werden. Verglichen mit einem CPU-Löser können die Berechnungen auf diese Weise um wenigstens eine Größenordnung beschleunigt werden. Vergleichbare Anwendungsfälle finden sich in unterschiedlichsten Bereichen. Sie reichen von simulierten Farbverläufen auf Papier [60], der Ausbreitung von Schadstoffen innerhalb von Städten in der Luft [228] über die Simulation von Fluidgemischen [321] und Seifenblasen [295] bis zur Simulation von Schmelz- und Fließphänomenen in Mehrphasenumgebungen [320] und des Schimmerns von Hitze und Luftspiegelungen [319]. Mit der Verfügbarkeit von CUDA und der einhergehend deutlich vereinfachten Programmierschnittstelle entwickelte sich ein allgemeiner Trend, GPUs zur Beschleunigung von numerischen Berechnungen einzusetzen. Zu den frühen Anwendungen auf dem Gebiet der Lattice-Boltzmann-Methode zählen insbesondere die Arbeiten von Tölke et al. [279, 281]. In diesem Zusammenhang ist auch ein prototypartiger Vorläufer von VIRTUALFLUIDS INTERACTIVE zur interaktiven Simulation zweidimensionaler Strömungen zu nennen, der von Linxweiler, Tölke und Krafczyk beschrieben wird [167]. Die im Folgenden beschriebene Implementierung für Strömungen im dreidimensionalen Raum setzt auf dem von Tölke [281] vorgeschlagenen Vorgehen auf.

5.3.1 Gridlayout

Der Relaxationsschritt stellt den numerisch aufwändigen Anteil des Lattice-Boltzmann-Verfahrens dar. In jedem Zeitschritt werden nahezu identische Kollisionsoperationen an allen Knoten des Berechnungsgitters ausgeführt. Ein nahe liegendes Vorgehen bei der Übertragung des Verfahrens auf die GPU besteht folglich darin, die Kollision in einer *Kernel*-Funktion auszuführen, die für jeden einzelnen Gitterknoten aufgerufen wird. Analog zu dem dreidimensionalen Berechnungsgitter wird dafür ein *Grid* aus Threadblöcken definiert, bei dem das zweidimensionale *Grid* eine Ebene in Richtung der Y- und Z-Achse aufspannt, während die eindimensionalen Blöcke entlang der X-Achse verlaufen (vgl. Abbildung 5.11). Die Dimensionen entsprechen dabei in allen drei Richtungen den Ausdehnungen des Berechnungsgitters. Aus dieser Wahl ergeben sich einige Randbedingungen für die Dimensionierung des Berechnungsgitters. Die Programmierumgebung erfordert zunächst keine Einschränkungen hinsichtlich der Größe des *Grids*, weshalb die Ausdehnung in Y- und Z-Richtung frei gewählt werden kann. Die Z-Richtung wird folglich als Strömungsrichtung des virtuellen Windkanals festgelegt und weist somit im Allgemeinen die größte Knotenanzahl auf. Für die X-Richtung ergibt sich eine Limitierung der Knotenanzahl durch die maximal mögliche Blockgröße. Je nach eingesetzter Hardware beträgt diese entweder 512 (*Compute Capability*[5] ≤ 1.3) oder 1024 *Threads* (*Compute Capability* > 1.3). Threadblöcke werden nach der Zuweisung zu einem Streaming-Multiprozessor für die Ausführung in sogenannte *Warps* partitioniert. Die Größe eines Warps entspricht bei heutigen Karten im Allgemeinen 32 *Threads*. Für eine effiziente Ausführung sollte die Blockgröße einem Vielfachen der *Warp*-Größe entsprechen. Ist dies nicht der Fall, werden fehlende *Threads* durch das sogenannte

[5] Die *Compute Capability* spezifiziert den Funktionsumfang einer GPU. Im Allgemeinen nimmt die *Compute Capability* mit fortschreitender Hardwaregeneration zu [239].

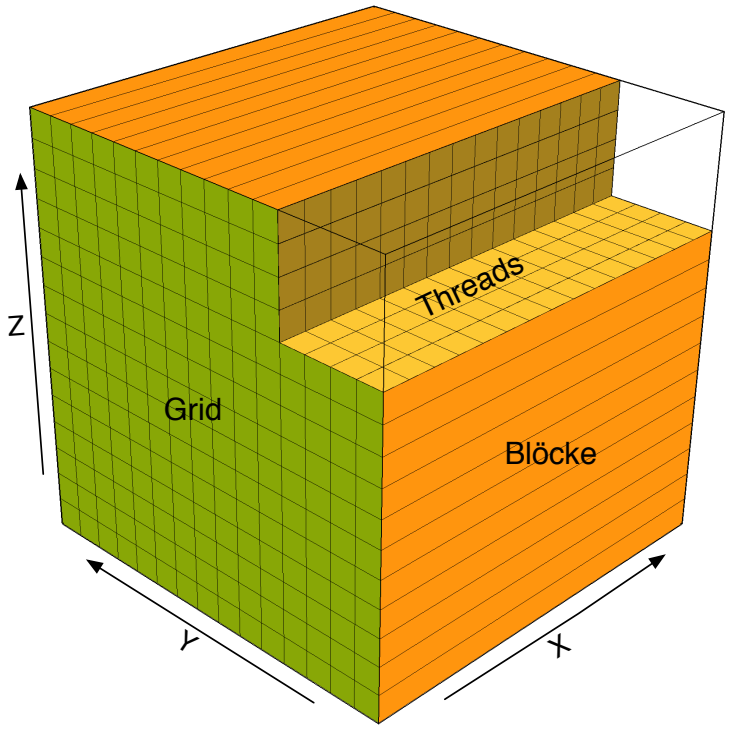


Abbildung 5.11: *Abbildung eines dreidimensionalen uniforme Berechnungsgitter auf ein CUDA-Grid (bestehend aus Threadblöcken).*

Padding bis zu einem notwendigen Vielfachen von 32 ergänzt und die verfügbaren Ressourcen somit nicht voll ausgeschöpft.

5.3.2 Datenlayout und Speicherzugriff

Im Kollisionsschritt werden die Lattice-Boltzmann-Verteilungen relaxiert, die wegen ihres großen Speicherbedarfs im globalen Speicher vorgehalten werden. Nach erfolgter Kollision werden die neuen Verteilungen zu den Nachbarknoten propagiert. Um unnötige Speicherbewegungen zu vermeiden, werden der Kollisions- und Advektionsschritt kombiniert in einer *Kernel*-Funktion durchgeführt. Dies gilt insbesondere für Zugriffe auf den globalen Speicher, der mit einer verhältnismäßig geringen Bandbreite angebunden ist. Dabei beeinflusst die Datentransferrate in erheblichem Maße die Gesamtleistung des Verfahrens, da das LB-Verfahren von Natur aus durch die Speicherbandbreite limitiert ist. Die erzielbare Transferrate wird dabei in erheblichem Maße durch das Zugriffsmuster bestimmt [143, 207, 208, 239] und kann im ungünstigsten Fall bis zu einer Größenordnung variieren [207]. Für einen effizienten Speicherzugriff müssen daher abhängig von der Hardwaregeneration verschiedene Anforderungen berücksichtigt werden. Zur Veranschaulichung dieser Restriktionen ist es hilfreich, den globalen Speicher in Form von Segmenten zu betrachten, die jeweils aus N Elementen bestehen. Bei aktuellen Hardwaregenerationen gilt $N = 16$ und entspricht damit der Größe eines halben *Warps*. Im günstigsten Fall kann ein Speicherzugriff auf eine einzelne Schreib- oder Lesetransaktion reduziert werden. Grundsätzlich ist dies jedoch nur dann möglich, wenn die *Threads* eines halben *Warps* lediglich Elemente zweier aufeinander folgender Segmente des globalen Speichers adres-

sieren. Bei GPUs mit einer *Compute Capability* ≤ 1.1 besteht zusätzlich die Einschränkung, dass der k-te Thread eines *Warps* auf das k-te Element des Segments zugreifen sollte, da andernfalls nur $\frac{1}{16}$ der möglichen Bandbreite genutzt wird.

Um diesen Anforderungen zu genügen, werden die Verteilungen in jeder Richtung jeweils auf einen eindimensionalen Vektor im globalen Speicher abgebildet. Die Indizierung der Verteilung innerhalb der kompakten Form erfolgt dabei entsprechend der Laufrichtung der Blöcke. Der am schnellsten variierende Index entspricht somit der X-Richtung. Da jedem *Thread* seine Position innerhalb des Blocks sowie dessen Postion im übergeordneten *Grid* bekannt ist, kann mit Hilfe dieser Informationen jedem *Thread* ein Knoten bzw. Verteilungssatz im globalen Speicher zugeordnet werden. Die Zuordnungsvorschrift entspricht dabei folgendem Algorithmus:

Algorithmus 5.1 Indexberechnung

```
x = threadIdx.x                          ▷ Index des Threads
y = blockIdx.x                           ▷ X-Index des Blocks
z = blockIdx.y                           ▷ Y-Index des Blocks
sizeX = blockDim.x                       ▷ Anzahl der Elemente in X-Richtung (Blockgröße)
sizeY = gridDim.x                        ▷ Anzahl der Elemente in Y-Richtung
index = sizeX*(sizeY*z + y) + x          ▷ Index im Vektor
```

Auf diese Weise ergibt sich ein Speicherlayout, das einen Lesevorgang der Verteilungen zu Beginn einer Kollision auf Grund der ausschließlich lokalen Speicherzugriffe mit der vollen Bandbreite ermöglicht. Dabei wird von jedem Thread ein vollständiger Verteilungssatz gelesen, wobei der k-te Thread eines *Warps* das k-te Element eines Speichersegments adressiert. Der Speicherzugriff eines halben *Warps* beschränkt sich demnach auf ein einzelnes Speichersegment. Dieses streng lokale Zugriffsmuster kann jedoch in der nachfolgenden Advektion nicht aufrecht erhalten werden, da die relaxierten Verteilungen zu den Nachbarknoten propagiert werden. Abbildung 5.12 veranschaulicht diese Situation anhand der Propagationsrichtungen in der X-Y-Ebene. Es wird verdeutlicht, dass eine Advektion in Y-Richtung mit den Anforderungen an einen performanten Zugriff konform ist. (Gleiches gilt analog für die nicht dargestellte Z-Richtung.) Besitzt die Propagationsrichtung hingegen eine X-Komponente, resultiert dies in einem um jeweils ein Element versetzten Speicherzugriff, in Folge dessen ein halber *Warp* zwei statt einem Speichersegment adressiert. Aus dem versetzen Zugriffsmuster ergeben sich abhängig von der jeweiligen GPU-Generation Einschränkungen hinsichtlich des effektiven Datendurchsatzes. An dieser Stelle wird ein erheblicher Vorteil GPUs aktueller Generation offensichtlich, da die Architektur auf Grund eines CPU-artigen Caches robust auf inkohärente Speicherzugriffe reagiert. Dabei werden globale Speicherzugriffe in einem schnellen Cache gepuffert, so dass der Datendurchsatz bei dem betrachteten Zugriffsmuster nahezu unbeeinflusst bleibt. Im Gegensatz dazu führt das versetzte Zugriffsmuster sowohl bei der G80 als auch bei der etwas moderneren GT200-Generation zu einem unter Umständen erheblich verminderten Datendurchsatz. Abbildung 5.13 veranschaulicht die effektive Durchsatzrate in Abhängigkeit vom Versatz des Speicherzugriffs. Es wird ersichtlich, dass die volle Bandbreite nur bei einem Versatz von null oder einem Vielfachen der halben *Warp*-Größe (16) erzielt wird. Andernfalls reduziert sich der Datendurchsatz um eine konstante Größe. Dieses Verhalten erklärt sich dadurch, dass globale Speichertransaktionen nicht auf Basis einzelner Elemente, sondern segmentweise erfolgen. Dabei werden Zugriffe der

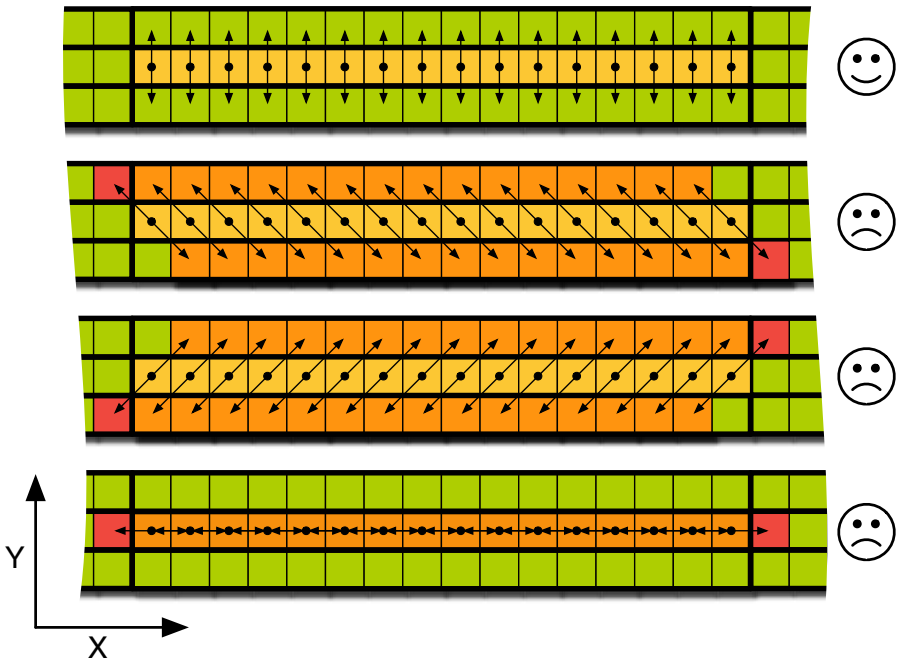


Abbildung 5.12: *Zugriffsmuster auf den globalen Speicher.*

Threads eines halben *Warps* gegebenenfalls zu einer Transaktion zusammengefasst. Adressieren alle *Threads* eines halben *Warps* ausschließlich Elemente des selben Speichersegments, resultiert dies in einer einzelnen Transaktion der Größe des Segments. Wie bereits erläutert, erfordern Propagationsrichtungen, die eine X-Komponente aufweisen, einen Speicherzugriff mit einem Versatz von einem Element. Deshalb werden von einem halben *Warp* zwei Segmente adressiert. In diesem Fall entspricht auch die Transaktionsgröße der Anzahl der Elemente von zwei Segmenten. Folglich entsteht ein Überhang zu kommunizierender Daten, weshalb die effektive Datendurchsatzrate von 120 GB/s auf etwa 70 GB/s absinkt. Noch gravierender ist der Leistungseinbruch bei Verwendung einer GPU der G80-Serie, da diese Architektur nur dann einen optimalen Datentransfer erzielt, wenn jeder k-te *Thread* eines halben *Warps* ausschließlich auf das k-te Element eines einzelnen Speichersegments zugreift. Nur dann können die Speicherzugriffe unterschiedlicher *Threads* eines halben *Warps* zu einer einzelnen Transaktion zusammengefasst werden. Ein versetztes Zugriffsmuster bedingt hingegen eine separate Speichertransaktion für jeden einzelnen *Thread*, so dass die Durchsatzrate in Folge des immensen Kommunikationsüberhangs im Fall einer GTX 8800 von 74 GB/s auf etwa 7 GB/s einbricht.

Das von Tölke [281] vorgeschlagene Vorgehen zur Vermeidung dieses Leistungseinbruchs sieht vor, den ursächlichen Versatz im eigentlich kohärenten Zugriffsmuster mit Hilfe des gemeinsam genutzten Speichers (*Shared Memory*) auszugleichen. Dieser sehr schnell angebundene Speicher steht wie bereits erläutert allen *Threads* innerhalb eines Blocks zur gemeinsamen Nutzung zur Verfügung. Da Grid- und Speicherlayout so gewählt wurden, dass die Ausrichtung der Threadblöcke identisch ist mit der Richtung des Versatzes im Speicherzugriff, ist es anschaulich formuliert möglich, die Bewegung der Verteilungen in X-Richtung zunächst im gemeinsam genutzten Speicher durchzuführen und diese anschließend ohne Versatz in den globalen Speicher zu schreiben. Um die Konsistenz der Daten zu gewährleisten, muss jedoch dafür Sorge getragen werden, dass der Zugriff verschiedener *Threads* auf

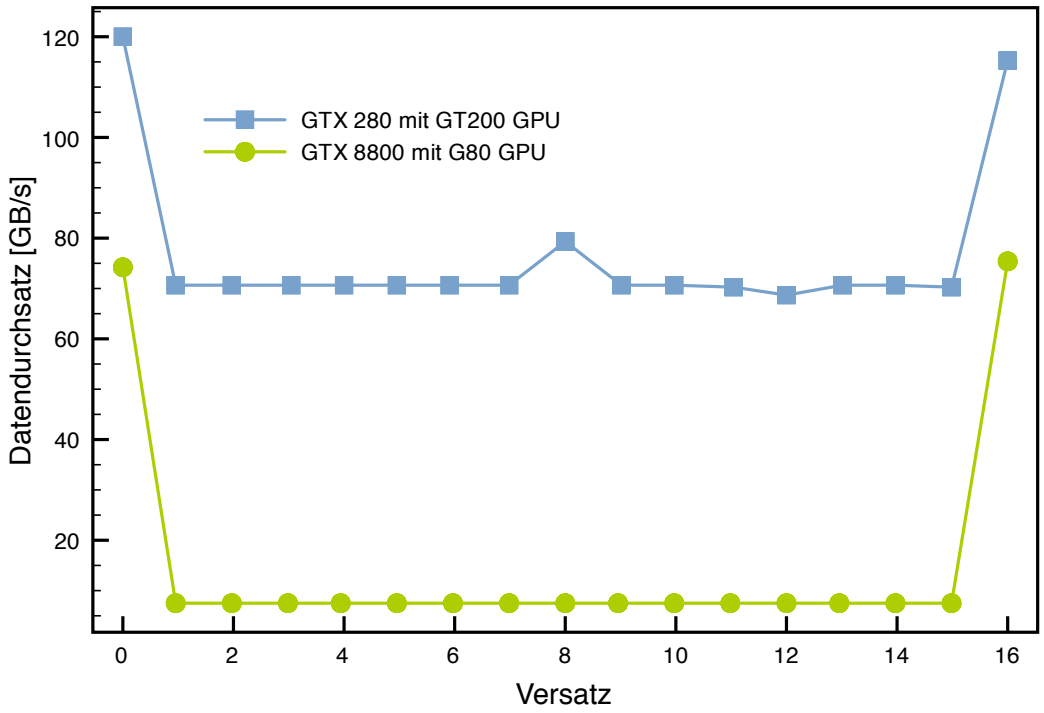


Abbildung 5.13: *Effektiver Datendurchsatz bei versetztem Zugriff auf den globalen Speicher basierend auf [207].*

den gemeinsamen Speicher synchronisiert erfolgt, wodurch wiederum geringe Leistungseinbußen entstehen. Auf einer GT200-Architektur trägt das Verfahren daher hinsichtlich der Gesamtleistung zu keiner Steigerung bei. Als Nachteil des Verfahrens sei erwähnt, dass aus der geringen Größe des gemeinsam genutzten Speichers insbesondere bei LB-Modellen mit großer Anzahl an Verteilungsrichtungen eine weitere Limitierung der Blockgröße resultiert.

Generell kann der schnell angebundene gemeinsam genutzte Speicher (*Shared Memory*) mit einem vom Entwickler verwalteten Cache verglichen werden. Dieser Speicher wird bei GPUs aktueller Generation automatisiert zum Puffern von Speicherzugriffen eingesetzt (L1- und L2-Cache), weshalb es seitens des Entwicklers nicht notwendig ist, versetzte Speicherzugriffe manuell auszugleichen.

5.3.3 EsoStripe - Effizienter Speicherzugriff durch optimiertes Datenlayout

Wie bereits in Abschnitt 5.1 erläutert, unterscheidet sich die Lattice-Boltzmann-Methode von anderen Finite-Differenzen-Verfahren dahingehend, dass die notwendigen Eingangsdaten für den Berechnungsschritt an jedem Knoten vollständig lokal vorliegen. Es ist daher grundsätzlich möglich, die Daten des vorangegangenen Zeitschritts zu überschreiben ohne die Konsistenz des Verfahrens zu beeinträchtigen. In der praktischen Anwendung bedeutet dies in der Regel jedoch, dass Kollision und Propagation unabhängig voneinander in zwei separaten Schritten durchgeführt werden. Hinsichtlich der zu erzielenden Leistung des Verfahrens bringt dieses Vorgehen jedoch eine merkliche Einschränkung mit sich, da alle Verteilungen zweifach gelesen und geschrieben werden. Es wird daher im Allgemeinen ein Ansatz verfolgt, der Kollision und Propagation in einem Schritt kombiniert.

Um dennoch die Konsistenz zu gewährleisten, werden dabei meistens alte und neue Verteilungen in einem doppelten Verteilungssatz vorgehalten, so dass bei großen Problemen ein erheblicher Anteil des Datenspeichers zu Gunsten der Rechenleistung verloren geht. Im Allgemeinen steht auf Grafikkarten jedoch erheblich weniger Speicher zur Verfügung als auf dem *Host*. Während *Host*-Speicher verhältnismäßig preiswert ist und leicht bis zu 144 GB betragen kann, können aktuelle Fermi-Karten architekturell bedingt maximal vergleichsweise geringe 6 GB Speicher adressieren. Diese Speichermenge steht allerdings nur auf den sehr teuren Tesla- und Quadro-Karten zur Verfügung. Aktuelle *Consumer*-Karten sind mit nicht mehr als 1,5 GB ausgestattet, weshalb eine defensive Nutzung des Speichers angeraten ist. Auf Grund der teilweise geringen Speicherausstattung kann dabei gegebenenfalls auch eine Reduktion des Speicherbedarfs zu Lasten der Komplexität des Verfahrens akzeptiert werden. So beschreiben Mattila et al. [184] eine Technik, die eine Kombination aus Kollisions- und Advektionsschritt erlaubt und dennoch eine einzelne Datenrepräsentation verwendet. Sie können zeigen, dass das Verfahren auf CPUs gleichzeitig mit einem Performancezuwachs einher geht [185]. Baily et al. haben diesen Ansatz auf GPUs übertragen [15]. Im Rahmen dieser Arbeit wird ein ähnlicher Ansatz verfolgt, der zudem die Eigenschaft eines ausschließlich segmentweisen Speicherzugriffs in der Propagation aufweist [102]. Dabei kann die Aktualisierung der Knoten grundsätzlich in einer beliebigen Reihenfolge oder auch vollständig simultan erfolgen. Die Konsistenz der Daten ist dennoch stets gewährleistet.

Damit mehrere Knoten simultan bearbeitet werden können, muss sichergestellt werden, dass jeder Knoten ausschließlich seine eigenen Eingangsdaten überschreibt. Das kann zum Beispiel dadurch erreicht werden, dass anstelle der eigentlichen Speicherbewegung lediglich Verweise auf die Speicherbereiche aktualisiert werden. Bei großen Systemen wäre die Folge dieses Vorgehens, dass die Daten eines Knotens mit jeder Aktualisierung zunehmend über den Speicher verteilt und die Operationen hochgradig nicht-lokal würden. Grundsätzlich ist es jedoch auch bei Eingrenzung auf die Eingangsdaten möglich, Speicherbewegungen durchzuführen, wenn man gestattet, dass ein Datum des Eingangsdatensatzes an eine andere als die ursprüngliche Position des Datensatzes geschrieben werden darf. Zur Veranschaulichung des grundlegenden Verfahrens wird zunächst der eindimensionale Fall betrachtet. Generell existiert für jede Verteilung des Lattice-Boltzmann-Verfahrens mit Ausnahme der sogenannten Ruheverteilung eine Verteilung mit entgegengesetztem Geschwindigkeitsvektor. Beispielhaft seien hier die Ost- (E) und Westverteilung (W) betrachtet. Während die Ostverteilung vom Standpunkt eines Knoten aus betrachtet aus dem Westen kommend nach Osten wandert, verhält sich die Westverteilung genau entgegengesetzt. Diese Bewegung der Verteilungen wird auf den Speicher dadurch abgebildet, dass die Ost- und Westverteilungen im Anschluss an einen Berechnungsschritt vertauscht werden. Der detaillierte Ablauf lässt sich ohne Beschränkung der Allgemeingültigkeit wie folgt beschreiben:

Die Ostverteilung eines Knotens an der Position x wird in einem Vektor $\mathscr{E}$ an der Position x gespeichert, wohingegen die Westverteilung in einem zweiten Vektor $\mathscr{W}$ an der Position $x+1$ vorgehalten wird.

$$f_E = \mathscr{E}[x] \tag{5.15}$$

$$f_W = \mathscr{W}[x+1] \tag{5.16}$$

Nach erfolgter Kollision werden die Verteilungen in entgegengesetzter Richtung gespeichert. Die Post-Kollisionsverteilungen sind in nachfolgender Notation mit einem Stern $(*)$ markiert.

$$\mathscr{W}[x+1] = f_E^* \tag{5.17}$$
$$\mathscr{E}[x] = f_W^* \tag{5.18}$$

Anschließend liegen die Ostverteilungen im Westspeicher und die Westverteilungen analog in dem ursprünglichen Ostspeicherbereich. Nach Kollision aller Knoten werden abschließend die Verweise auf Ost- und Westvektoren vertauscht. Die tatsächliche Speicherbewegung resultiert dabei aus dem versetzten Index in einer der zwei entgegengesetzten Richtungen. Der Versatz entspricht in diesem Fall 1 und es wird eine Zählweise von West nach Ost angenommen. Dementsprechend werden die Westverteilungen eines Knotens von dem östlichen Nachbarknoten gelesen, während die Ostverteilungen lokal vorliegen. Nach der Kollision werden die Westverteilungen hingegen lokal geschrieben während die Ostverteilungen zu den Nachbarknoten geschrieben werden. Durch den Austausch der Verweise auf die jeweiligen Datenspeicher der Ost- und Westrichtungen resultiert anschließend die tatsächliche Speicherbewegung der Verteilungen an den entsprechenden Knoten. Diese Methode kann leicht für beliebige Verteilungsrichtungen generalisiert werden. Dazu ist es lediglich notwendig, die Verteilungsrichtungen nach jedem LB-Schritt mit der entgegengesetzten Richtung zu tauschen. Für jede Verteilung C mit der Bewegungsrichtung $\vec{c}$ und die entgegengesetzt gerichtete Verteilung $\hat{C}$ mit der Bewegungsrichtung $\vec{\hat{c}} = -\vec{c}$ erfolgt der Speicherzugriff dabei entsprechend folgendem Schema:

$$f_C = \mathscr{C}[x,y,z] \tag{5.19}$$
$$f_{\hat{C}} = \hat{\mathscr{C}}[x+c_x, y+c_y, z+c_z] \tag{5.20}$$

Für das Zurückschreiben der relaxierten Verteilungen gilt analog:

$$\hat{\mathscr{C}}[x+c_x, y+c_y, z+c_z] = f_C^* \tag{5.21}$$
$$\mathscr{C}[x,y,z] = f_{\hat{C}}^* \tag{5.22}$$

Abbildung 5.14 veranschaulicht das Verfahren am zweidimensionalen Anwendungsbeispiel.

Auf Grund seiner Eigenart die Verteilungen implizit durch einen Austausch der Verteilungsrichtung zu propagieren, wurde das Verfahren von Geier et al. auf den Namen *Esoteric Twist* oder kurz *EsoTwist* getauft. Eine Besonderheit von *EsoTwist* ist das Verhalten von Knoten, die keine Aktualisierung erfahren. Werden die Verteilungen eines Knotens nicht relaxiert, so erfolgt durch den Austausch der Verteilungsvektoren ein implizites Invertieren der Verteilungsrichtungen, die sich somit analog zu einer knotenbasierten *Bounce-Back*-Randbedingung verhält. Knoten die eine Haftrandbedingung abbilden bedürfen folglich keiner besonderen Behandlung.

Neben einer Reduktion auf einen einzelnen Verteilungssatz ermöglicht das *EsoTwist*-Verfahren in einer Variation zusätzlich einen vollständig segmentweisen Zugriff auf den globalen Speicher, so dass

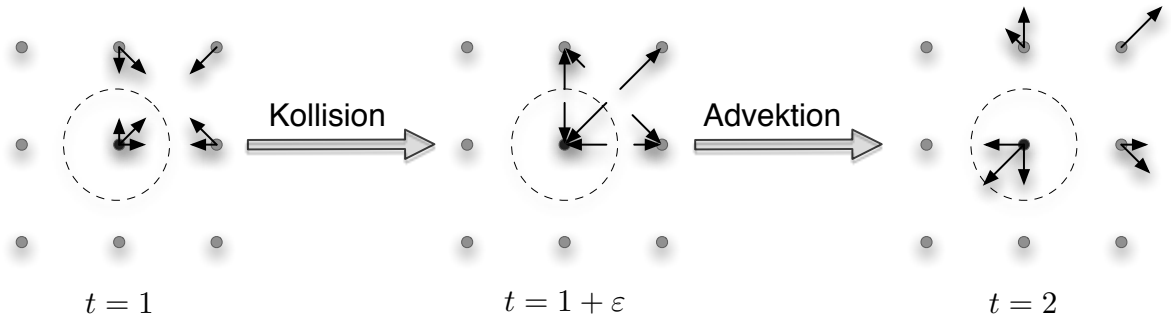


Abbildung 5.14: *Veranschaulichung des EsoTwist-Verfahrens am Beispiel des D2Q9-Models. Entgegengesetzte Verteilungen eines Knotens liegen zur einen Hälfte lokal und zur anderen Hälfte an benachbarten Knoten vor. Nach erfolgter Kollision werden die Verteilungen in der jeweils entgegengesetzten Richtung gespeichert. Die Advektion erfolgt anschließend durch den Austausch der Verweise auf die jeweils entgegengesetzten Verteilungen.*

die Propagation auch auf älteren GPU-Generationen ohne die Verwendung von *Shared Memory* auskommt. Im dreidimensionalen Anwendungsfall erfolgt dazu in Laufrichtung der *Threads*, die in diesem Fall mit der X-Richtung korrespondiert, eine geometrische Transformation der Knotenanordnung im Speicher, während in den übrigen beiden Richtungen das übliche *EsoTwist*-Verfahren angewendet wird. Die Länge des Gebiets in X-Richtung entspricht dabei L_x. Für diese Länge gilt nach wie vor die Einschränkung, einem Vielfachen der halben *Warp*-Größe (16) zu entsprechen. L_x wird in n Teile (*Slices*) der Größe $L = 16$ untergliedert. Für jeden Knoten kann der jeweilige *Slice* durch folgende Berechnungsvorschrift ermittelt werden:

$$\text{slice}(x) = \left\lfloor \frac{x}{L-1} \right\rfloor \tag{5.23}$$

Durch Verwendung des Integerdatentyps kann bei dieser Operation ein ganzzahliger Index garantiert werden. Durch eine explizite Typumwandlung des Ergebnisses in einen ganzzahligen Wert erfolgt gleichzeitig das erforderliche Abrunden. Ist für einen Knoten der entsprechende *Slice* ermittelt, kann die Position des Knotens innerhalb des *Slices* bestimmt werden. Dieser Index wird im Folgenden als *Stripe* bezeichnet und wie folgt bestimmt:

$$\text{stripe}(x) = x - \text{slice}(x)\,n \tag{5.24}$$

Für die Abbildung der Informationen eines Knotens auf den Speicher werden *Slices* und *Stripes* schließlich getauscht. Die Speicherposition $\mathscr{X}$ eines Knotens an der Stelle x ergibt sich somit wie folgt:

$$\mathscr{X} = \text{stripe}(x)\,L + \text{slice}(x) \tag{5.25}$$

Werden die Informationen eines Knotens entsprechend diesem Schema im globalen Speicher vorgehalten, so liegen benachbarte Knoten in X-Richtung im Abstand von genau einer Speichersegmentgröße. Demzufolge entspricht der Versatz in dem Advektionsschritt nunmehr zuverlässig einem Vielfachen der Segmentgröße, so dass alle Lese- und Schreibzugriffe mit dem optimalen Datendurchsatz

erfolgen. Diese Variation des *EsoTwist*-Verfahrens wird von Geier et al. [102] als *EsoStripe* bezeichnet, für das sich folgende Zuordnungsvorschrift ergibt:

$$f_C = \mathscr{C}[\mathscr{X}, y, z] \tag{5.26}$$
$$f_{\hat{C}} = \hat{\mathscr{C}}[\mathscr{X} + Lc_x, y + c_y, z + c_z] \tag{5.27}$$

$$\hat{\mathscr{C}}[\mathscr{X} + Lc_x, y + c_y, z + c_z] = f_C^* \tag{5.28}$$
$$\mathscr{C}[\mathscr{X}, y, z] = f_{\hat{C}}^* \tag{5.29}$$

Diese Gleichungen sind allgemein für alle bis auf den letzten *Stripe* gültig, da dieser im *EsoStripe*-Schema mit dem ersten *Stripe* kommuniziert. Dadurch impliziert das Verfahren periodische Randbedingungen. In diesem Fall lässt sich die Kontinuität des Speicherzugriffsschemas jedoch nicht vollständig aufrecht erhalten, da der Zugriff um ein Element versetzt erfolgt. Dennoch bleibt der Zugriff weiterhin auf ein einzelnes Speichersegment begrenzt. In Abbildung 5.16 ist die Leistung des *EsoStri-*

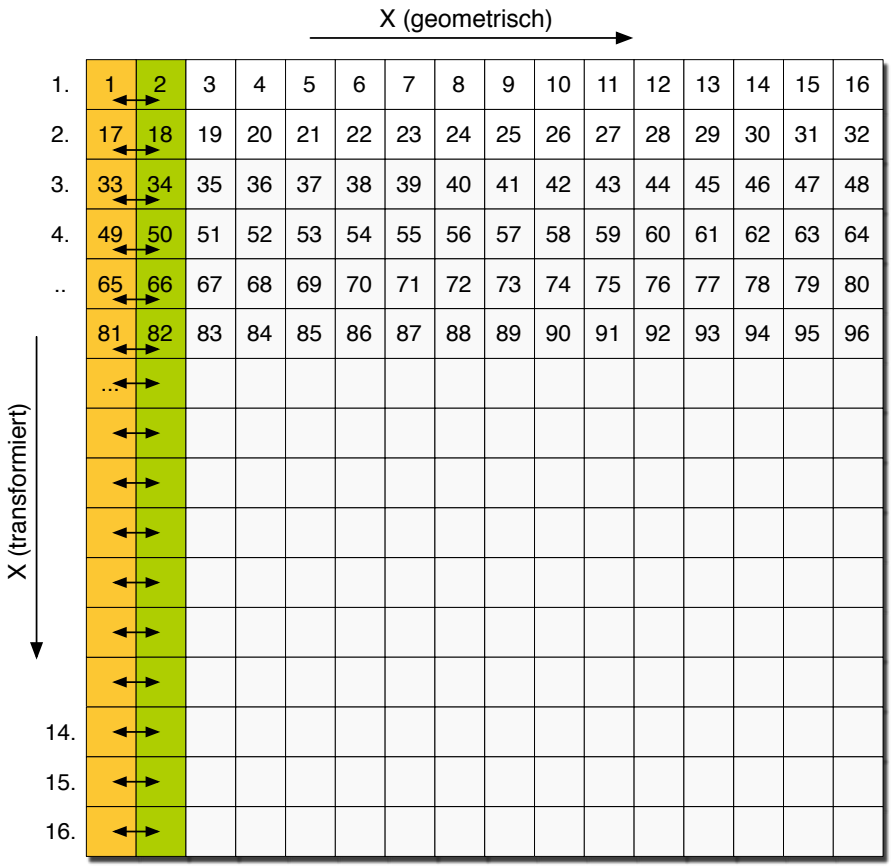


Abbildung 5.15: *Veranschaulichung der transformierten Speicherschemas. Die Kommunikation in X-Richtung erfolgt vollständig segmentweise.*

pe-Verfahrens den klassischen Speicherlayouts gegenüber gestellt. Dargestellt ist die erzielte Anzahl

an Knotenaktualisierungen pro Sekunde auf GPUs unterschiedlicher Generationen. Es wird ersichtlich, dass der *EsoStripe*-Ansatz auf aktuellen GPU-Generationen den übrigen Verfahren überlegen ist. Auf GPUs der G80-Serie hingegen führt der nicht sequentielle Datenaustausch zwischen dem letzten und dem ersten Stripe zu einem Leistungseinbruch. Die optimale Leistung kann auf dieser Architektur lediglich durch Verwendung des *Shared Memory* erzielt werden. Auf moderneren Chips kann hingegen kaum eine Leistungssteigerung erzielt werden.

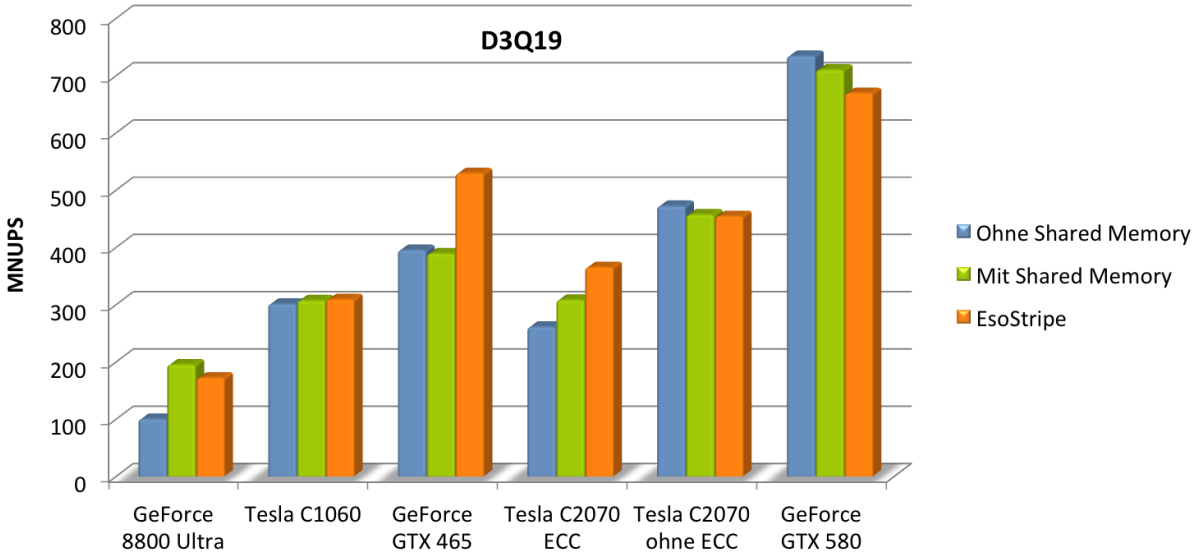


Abbildung 5.16: *Leistung von LB-Implementierungen auf GPUs unterschiedlicher Generationen. Die Grafik stellt die Anzahl der Knotenaktualisierungen pro Sekunde unterschiedlicher Verfahren auf GPUs unterschiedlicher Generationen dar. Die 8800 Ultra Karte ist mit einer G80 GPU ausgestattet, während in der Tesla C1060 ein GT200-Chip zum Einsatz kommt. Die GTX 465 hingehen ist eine* Consumer*-Karte der Fermi-Generation. Berechnet wurde ein Gebiet mit ca. 1 Mio. Knoten.*

5.3.4 Parallele Simulation unter Einsatz mehrerer GPUs

Die Architektur moderner Computer erlaubt es, mehrere Grafikkarten in einem einzelnen System zu betreiben. Aktuell sind Systeme mit vier GPUs keine Seltenheit. Beide großen Hersteller von Grafikprozessoren - NVIDIA und AMD - haben bereits Modelle vorgestellt, bei denen zwei Grafikprozessoren auf einer Karte verbaut sind, so dass in Zukunft mit einer zunehmenden Anzahl an GPUs in einem System gerechnet werden kann. Wird ein gewisser Mehraufwand bei der Entwicklung in Kauf genommen, können diese Ressourcen in Kombination für parallele Berechnungen eingesetzt werden. Durch die Verteilung des numerischen Problems auf mehrere Komponenten kann dabei einerseits ein Zuwachs in der Gesamtrechenleistung erwartet werden; andererseits erhöht sich die mögliche Systemgröße, da in Summe ein größerer globaler Speicher zu Verfügung steht. Im Hinblick auf interaktive Simulationen ist dabei insbesondere die Leistungssteigerung in Folge der Lastverteilung von Bedeutung. Im theoretischen Fall einer ideal skalierenden Parallelisierung auf N GPUs wird es somit möglich, bei gleichbleibender Simulationsgeschwindigkeit eine N-fache Systemgröße zu berechnen

oder alternativ die Simulationsgeschwindigkeit bei gleichbleibender Systemgröße um das N-fache zu vergrößern.

Für eine parallele Berechnung wird das Strömungsgebiet mit einem einfachen Partitionierungsansatz analog zu Abbildung 5.17 in N möglichst gleich große Teilgebiete zerlegt, von denen jedes für sich von einer der N GPUs bearbeitet wird. Die Simulation verläuft dabei in $N+1$ asynchronen Ausführungs-

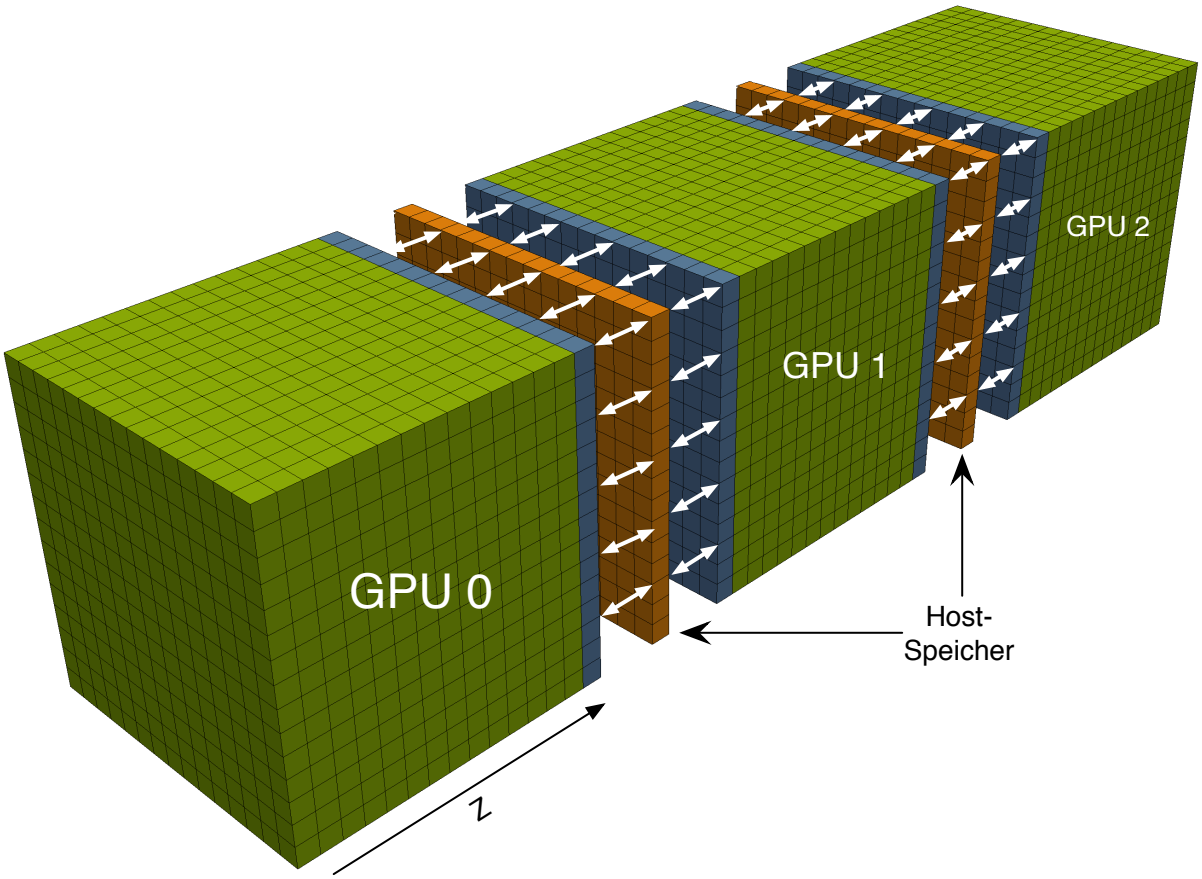


Abbildung 5.17: *Partitionierung des Strömungsgebiets zur Berechnung unter Verwendung von drei Grafikprozessoren.*

strängen, von denen einer für die Koordination vorgesehen ist, während die übrigen jeweils die isolierte Berechnung eines Teilgebiets unter Verwendung einer dedizierten GPU durchführen. Die strömungsbeschreibenden Verteilungen der jeweiligen Partition werden dabei in dem globalen Speicher der entsprechenden GPU vorgehalten. Eine besondere Schwierigkeit resultiert aus der notwendigen bidirektionalen Kommunikation zwischen den jeweiligen Randknoten (blau dargestellt) benachbarter Teilgebiete. Da das CUDA-Programmiermodell den Zugriff einer GPU auf den globalen Speicher eines anderen CUDA-fähigen Gerätes zum derzeitigen Zeitpunkt (noch) nicht unterstützt[6], ist es notwendig, dass diese Kommunikation unter Verwendung des *Host*-Speichers erfolgt. Der Speicher auf dem *Host* dient dabei als Puffer (orange dargestellt), in dem die jeweiligen abgehenden Verteilungen aus dem globalen Speicher der GPU zwischengespeichert werden, bevor sie wiederum in den globalen Speicher der benachbarten GPU kopiert werden können. Die Kommunikation der Randknoten

[6] Diese Funktion ist für die kommende CUDA Version 4.0 vorgesehen [210].

in Partitionierungsrichtung ist dabei bidirektional, weshalb der Austausch der Verteilungen in beiden Richtungen erfolgen muss. Der notwendige Datentransfer zwischen dem globalen Speicher auf der Grafikkarte und dem Arbeitsspeicher auf dem *Host* stellt dabei den limitierenden Faktor dar, da die Kommunikation über den PCI-Express-Bus verläuft, dessen maximale theoretische Datentransferrate mit 8 GB/s angegeben ist [143]. Im Vergleich dazu leistet die 384 Bit breite Anbindung des globalen Speichers an die GPU bei einer aktuellen Tesla C2070 theoretisch bis zu 144 GB/s [209]. Der Datenaustausch zwischen den Teilgebieten nimmt daher eine besondere Bedeutung im Hinblick auf die Gesamtperformance ein. In der Praxis hat sich gezeigt, dass eine Datentransferrate von mehr als 5 GB/s unter Verwendung des Speicherdirektzugriffs (DMA) (vgl. Abschnitt 5.2.3) erzielt werden kann [86, 207]. In jedem Fall ist es erforderlich, dass die zu kommunizierenden Daten zusammenhängend im Speicher abgelegt sind. Aus diesem Grund erfolgt die Gebietszerlegung in Z-Richtung, da die Randknoten somit zusammenhängend in einer X-Y-Ebene liegen. Der größte Leistungsgewinn lässt sich jedoch dadurch erzielen, dass die Kopieroperationen der Randknoten von den Berechnungen an den übrigen inneren Knoten überdeckt werden. Dieses als *Communication Hiding* bezeichnete Vorgehen wird von CUDA durch Verwendung sogenannter *Streams* unterstützt. Ein CUDA-*Stream* repräsentiert eine Sequenz von Operationen, die konsekutiv ausgeführt werden. Mögliche Operationen eines *Streams* sind beispielsweise Kernelaufrufe oder Kopiervorgänge zwischen globalem Speicher auf dem *Device* und dem *Host*-Speicher. Bei Verwendung mehrerer *Streams* werden die jeweiligen Sequenzen asynchron ausgeführt. Wenngleich zum heutigen Zeitpunkt die wenigsten GPUs simultan mehrere *Kernel*-Funktionen ausführen können, so unterstützen nahezu alle Karten (Compute Capability ≥ 1.1) während der Ausführung von *Kernel*-Funktionen einen nebenläufigen Datentransfer zwischen *Host* und *Device*. Wird nun die Berechnung der Randknoten und die der inneren Knoten auf unterschiedliche Kernelaufrufe verteilt, so kann simultan zur Ausführung der Operationen auf den inneren Knoten die Kommunikation zwischen den Randknoten erfolgen. Bei hinreichend großen Teilgebieten kann auf diese Weise die notwendige Kommunikation zwischen den Randknoten vollständig im Hintergrund erfolgen ohne zusätzliche Zeit zu beanspruchen. Ausgehend von einer effektiven Datentransferrate von 5 GB/s des PCI-Express-Busses und Anbindung der GPU an den globalen Speicher von effektiv 100 GB/s gilt für die Teilgebietslänge in Z-Richtung L_Z überschlägig [280]:

$$L_Z \geq \frac{\frac{100\,GB/s}{5\,GB/s}}{2} = 40 \tag{5.30}$$

In Folge notwendiger Kontroll- und Synchronisierungsvorgänge zwischen den einzelnen Ausführungssträngen ist eine Beeinflussung der Gesamtrechenleistung jedoch grundsätzlich nicht zu vermeiden. Abbildung 5.18 veranschaulicht die Leistung des *EsoStripe*-Verfahrens unter Einsatz von bis zu vier Tesla C1060 Karten bei unterschiedlichen Längen der geschnittenen Z-Richtung. Aus der Darstellung geht unter anderem hervor, dass unter Verwendung von maximal vier Tesla C1060 eine Rechenleistung von bis zu 1440 Millionen Knotenaktualisierungen pro Sekunde (1440 MNUPS) erzielt werden kann. Die Bedeutung dieses Resultats wird insbesondere dann ersichtlich, wenn man es in Relation zu vergleichbaren Ergebnissen CPU-basierter Simulationen betrachtet. So berichten beispielsweise Wellein et al. [298] von einer effizienten Implementierung des D3Q19 Modells, das auf einem mit 3,4 GHz getakteten Intel Xeon Prozessor-Kern bei doppelter Genauigkeit eine serielle Performance von 7,8 MNUPS erzielt. In einer aktuellen Veröffentlichung stellen Feichtinger, Habich und andere [87] eine parallele Umsetzung vor, die bei einfacher Genauigkeit auf vier Xeon Kernen der X5570-Serie bis zu 38 MNUPS erreicht.

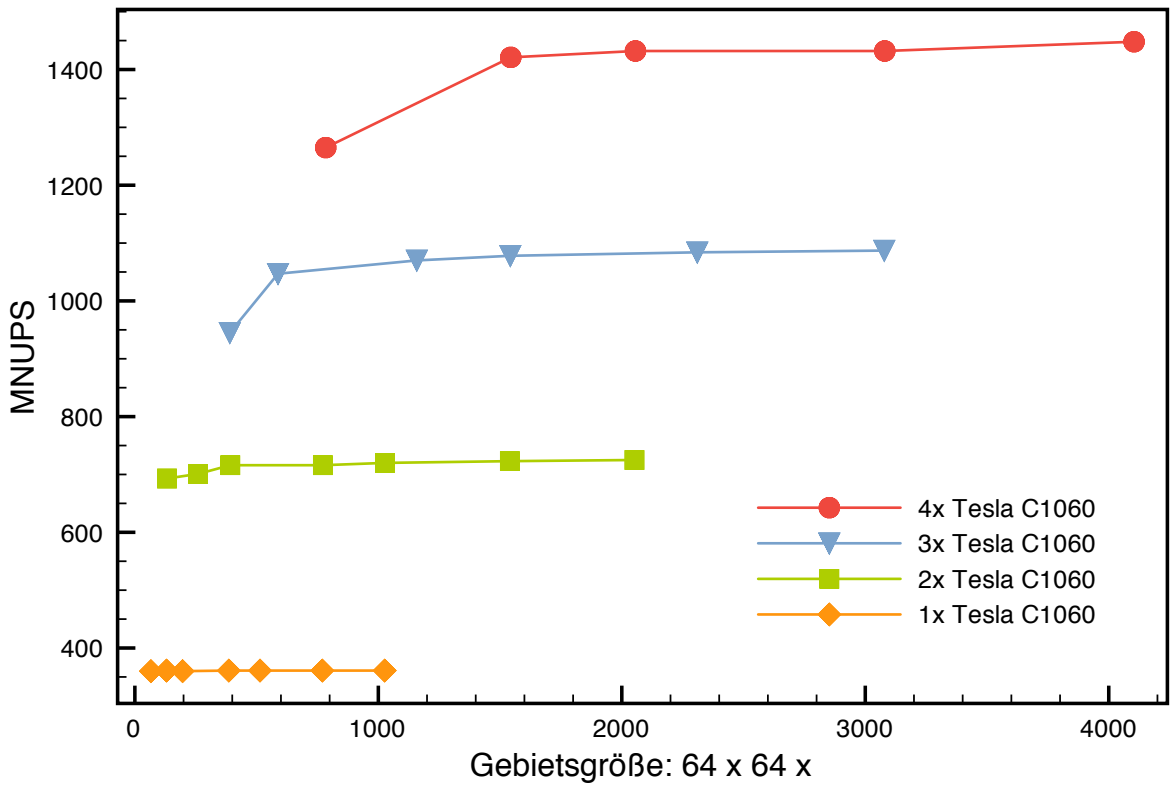


Abbildung 5.18: *Leistungsvergleich des EsoStripe-Verfahrens bei Verwendung meherer GPUs.*

5.4 Integration in die interaktive Umgebung

Generell erfordert eine effektive Steuerung und Analyse von Simulationen eine hohe Integration von interaktiver Umgebung und Berechnungskern [133, 216]. Allein dadurch wird einerseits eine effiziente Extraktion und anschließende Analyse der relevanten Ergebnisdaten möglich und gleichzeitig ein responsives Verhalten des Berechnungskerns auf Änderungen von Simulationsparametern erreicht. Die Integration erfordert in der Regel einige Änderungen am ursprünglichen Simulationsverfahren, die jedoch nach Möglichkeit keine Leistungseinbußen mit sich bringen sollten.

5.4.1 Kommunikation zwischen Rahmenanwendung und Berechnungskernen

Die *Computational Steering*-Rahmenanwendung definiert zur Integration von Lattice-Boltzmann-basierten Simulationskernen die Schnittstelle `LBSimulation`, die zur Spezifikation der Kommunikation zwischen den Komponenten dient. Dieses Vorgehen folgt dem *Abstract Server*-Muster [180], bei dem die Abhängigkeit des Konsumenten eines Dienstes (Rahmenanwendung) von dem Anbieter (Simulationskern) durch Einführung einer verbindlichen Schnittstelle auf Seiten des Konsumenten umgekehrt wird. Durch Anwendung des Prinzips der Umkehrung der Abhängigkeiten (DIP) an dieser Stelle kann vermieden werden, dass Änderungen an dem Simulationskern zusätzlich auch Anpassungen der Rahmenandwendung nach sich ziehen. Gleichzeitig wird auf diese Weise das *Open Closed-*

Prinzip (OCP) in Hinblick auf die Erweiterung der Anwendung um alternative Simulationsverfahren erfüllt. Neue Berechnungskernel müssen lediglich die Schnittstelle `LBSimulation` implementieren, um im Kontext von VIRTUALFLUIDS INTERACTIVE verwendet werden zu können. Da die Implementierung der Berechnungskerne auf CUDA-C basieren, die Schnittstelle der Rahmenanwendung jedoch die Sprache C++ erfordert, ist ein zusätzlicher Schritt für die Integration erforderlich. Um die Simulationskomponenten in Konformität mit der geforderten Schnittstelle zu bringen, werden die C-basierten Routinen unter Anwendung des *Adapter*-Musters in gesonderten Klassen gekapselt. Die Aufgabe der Adapterklassen besteht der Definition nach darin, die Schnittstelle der Komponenten an eine erwartete Schnittstelle anzupassen, und damit eine ansonsten ausgeschlossene Kommunikation mit den Klienten zu ermöglichen [97]. Um anderen Entwicklern eine Implementierungshilfe zur Verfügung zu stellen und gleichzeitig Redundanzen zu vermeiden, wird die Grundstruktur der Adapterklassen in Form der Basisklasse `GPUKernelAdapter` abstrahiert. Dem Entwurfsmuster *Template Method* [97] folgend, bietet die abstrakte Basisklasse ein Gerüst für die Integration eigener CUDA-basierter LB-Implementierungen. In VIRTUALFLUIDS INTERACTIVE stehen bereits Kernel für das D3Q13, D3Q19 und D3Q27-Modell zur Verfügung. Dabei können die Berechnungen jeweils wahlweise unter Verwendung einer oder mehrerer GPUs erfolgen. Abbildung 5.19 veranschaulicht die Integration der Berechnungskerne in die interaktive Umgebung.

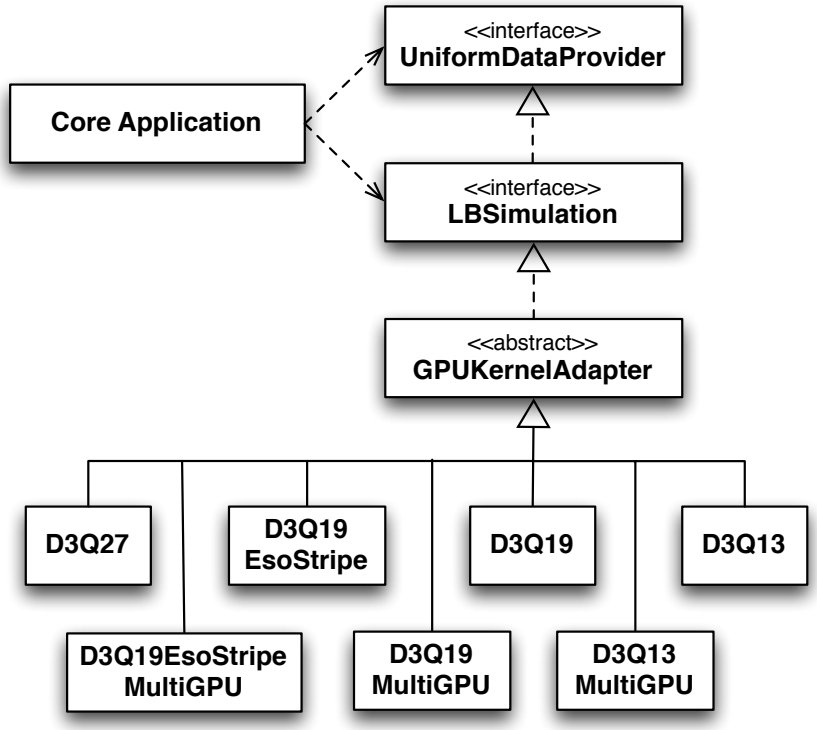


Abbildung 5.19: *Integration der Berechnungskerne in die interaktive Simulationsumgebung.*

5.4.2 Steuerungsparameter und Ergebnisdaten

Die einheitlich geltende Schnittstelle aller Simulationskerne legt zum einen die Steuerungsparameter fest und definiert zudem das Format der Geometriebeschreibung sowie der Ergebnisdaten. Die

Visualisierungskomponente der Rahmenanwendung verwendet zur Darstellung ausschließlich die Ergebnisdaten. Die 3D-Repräsentation wird dabei nicht unmittelbar zur Steuerung der Simulation eingesetzt, so dass die Komponente nicht von der Steuerungsfähigkeiten der Berechnungskerne abhängen sollte. Nach dem Prinzip der Aufteilung der Schnittstellen (ISP) wird daher eine weitere Schnittstelle (`UniformDataProvider`) extrahiert, die einen ausschließlichen Zugriff auf die Ergebnisdaten ermöglicht und die überflüssige Abhängigkeit entfernt (vgl. Abbildung 5.19). Die konventionelle Komponente der Benutzerschnittstelle basiert hingegegen auf den Steuerungsfähigkeiten und ermöglicht dem Anwender beispielsweise die Parametrisierung der Einflussrandbedingung und des Aktualisierungsintervalls sowie die Aktivierung des LES-Modells. Bei der Implementierung der Schnittstellen gilt es zu beachten, den Zugriff auf die Steuerungsparameter und auch die Ergebnisdaten zu synchronisieren, da die Berechnungen der Simulationskerne grundsätzlich nebenläufig zu der grafischen Benutzeroberfläche in einem separaten *CPU-Thread* ausgeführt werden. Auf diese Weise wird ein »Einfrieren« der Benutzeroberfläche in Folge der andauernden Berechnugen verhindert und eine responsive Interaktionsfähigkeit sichergestellt. Steuerungsparameter, Geometriebeschreibung und Ergebnisdaten liegen daher redundant sowohl auf dem *Host* als auch auf dem *Device* und werden nach einer Modifikation dieser Daten repliziert. Zu diesem Zweck wird die Berechungsschleife, in der kontinuierlich die Kollisions- und Propagationsoperationen ausgeführt werden, entsprechend Abbildung 5.20 um entsprechende Aktualisierungsroutinen erweitert. Einen besonderen Einfluss auf die Responsivität der Anwendung haben dabei die Aktualisierung der Geometriebeschreibung und der Ergebnisdaten, da in beiden Fällen größere Datenmengen zwischen Host und Device transferiert werden müssen. Um einen schnellstmöglichen Datentransfer zu ermöglichen, wird auch an dieser Stelle von der Möglichkeit des Speicherdirektzugriffs (DMA) mit einer effektiven Transferrate von bis zu 5 GB/s Gebrauch gemacht. Im Vergleich dazu können Ergebnisdaten in einer verteilten Umgebung, eine 1 GBit/s Netzwerkverbindung vorausgesetzt, in der Regel effektiv mit maximal 80 MB/s kommuniziert werden. Damit stellt der Transfer von Ergebnisdaten in den meisten verteilten *Computational Steering*-Anwendungen einen erheblichen Flaschenhals dar, der einen signifikanten Einfluss auf die Interaktivität der Anwendung hat. Aktuell kann bei CPUs und GPUs ein deutlich größeres Leistungswachstum beobachtet werden, als es bei der Bandbreite von Netzwerkanbindungen der Fall ist. In Folge dieser Entwicklung ist in naher Zukunft keine Änderung der Situation abzusehen. An dieser Stelle profitiert VIRTUALFLUIDS INTERACTIVE in erheblichem Maße von der signifikant höheren Bandbreite. Der Datentransfer wird zusätzlich durch eine Reduktion der Datenmenge beschleunigt. Anstatt der vollständigen Verteilungssätze werden dazu lediglich die makroskopischen Größen übermittelt, die in einem vorausgehenden Schritt (in situ) auf dem Device berechnet werden (vgl. Abschnitt 6.2.2). Dieses Vorgehen kommt einer semantischen Kompression gleich und geht mit einem Informationsverlust einher. Jedoch werden Anwender in der Regel weniger an den einzelnen Verteilungen als vielmehr an dem makroskopischen Verhalten des Strömungsprozesses interessiert sein. Bestehende Ergebnisdaten werden bei einer Aktualisierung ressourcenschonend überschrieben, so dass der Speicherbedarf der Anwendung zur Laufzeit konstant bleibt. Die Rahmenanwendung wird mit Hilfe des *Observer*-Musters über die aktualisierten Ergebnisdaten unterrichtet und aktualisiert daraufhin gleichfalls die Visualisierung.

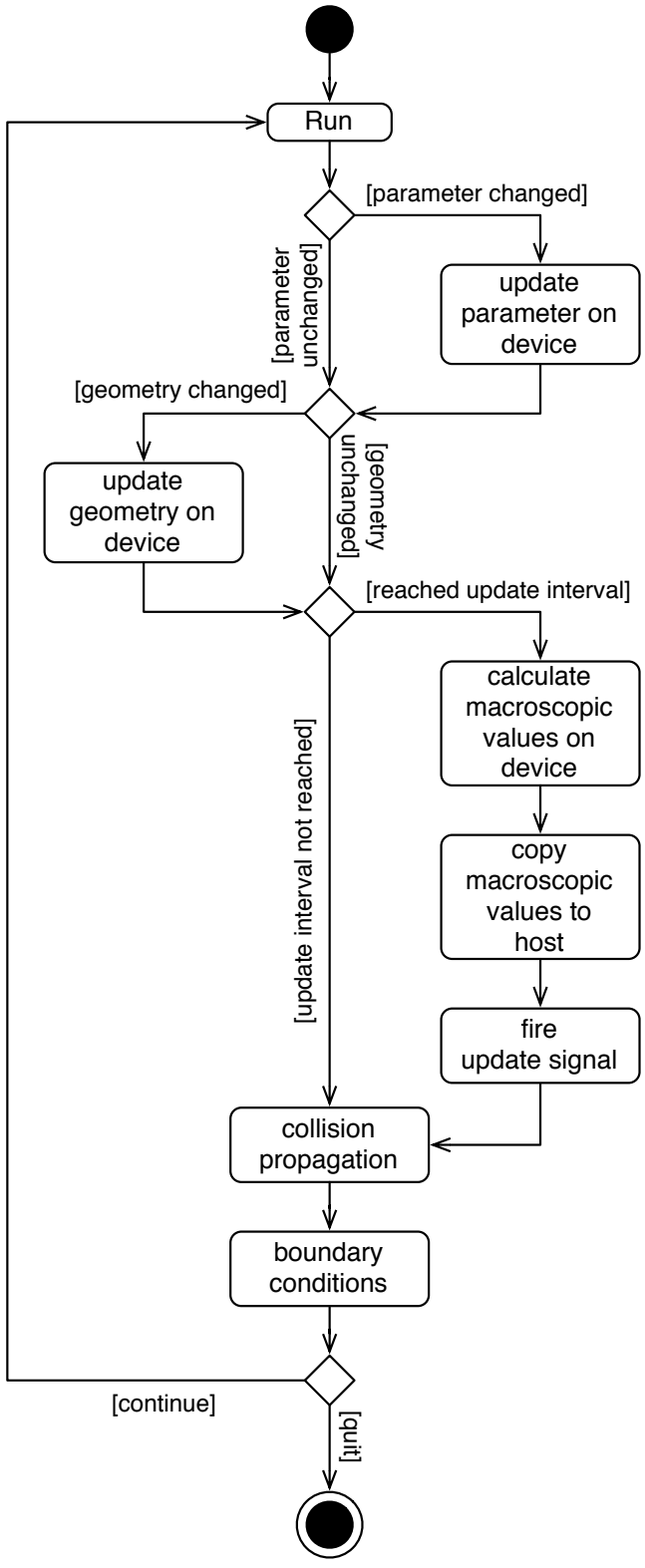


Abbildung 5.20: *Aktivitätsdiagramm der interaktiven Berechnungsschleife des Berechnungskerns.*

6 Das Visualisierungssystem

In diesem Kapitel wird die Bedeutung der Visualisierung für die Analyse von Ergebnisdaten verdeutlicht. Es wird ein Überblick über die in VIRTUALFLUIDS INTERACTIVE verwendeten Verfahren gegeben und ein komponentenbasierter Ansatz vorgestellt, der den Einsatz von Visualisierungstechniken im Rahmen eigener Erweiterungen erleichtert. Zudem werden Limitierungen klassischer Verfahren und Systeme im Zusammenhang mit der Verarbeitung CFD-typischer Massendaten diskutiert und Lösungsansätze vorgestellt.

6.1 Wissenschaftliche Visualisierung

Der Mensch ist ein visuell dominiertes Wesen, das seine Umwelt im überwiegenden Maße bildlich wahrnimmt [79]. Die sogenannte »visuelle Intelligenz« (visueller Cortex) nimmt mehr als die Hälfte der Großhirnrinde (Cortex cerebri) ein [123]. 75% aller Informationen der realen Welt werden visuell aufgenommen; lediglich 13% von den auditiven sowie die verbleibenden 12% von den übrigen Sinnen. Die visuelle Wahrnehmung ist eng verbunden mit den kognitiven Fähigkeiten, zu denen sowohl das Lernen und logische Denken als auch das Gedächtnis gehören. Der Begriff Kognition steht im wissenschaftlichen Sinne für die Aneignung und Anwendung von Wissen [64]. Bereits Aristoteles stellte fest:

> »Nihil est in intellectu quod non erat in sensu«[1]

Diese enge Relation zwischen Dingen, die wir sehen, und unseren Gedanken zeigt sich in vielen Metaphern des täglichen Lebens. Wenn wir uns beispielsweise jemanden vorstellen, der angestrengt geistig arbeitet, so denken wir möglicherweise an einen Wissenschaftler, der ein Diagramm zeichnet, während er ein aufgeschlagenes Buch als Informationsquelle neben sich hält. Oder wir stellen uns einen Börsenmakler vor, der angespannt zahlreiche Monitore mit Aktienkursen im Auge behält, um schnell auf aktuelle Ereignisse reagieren zu können. Diese Verflechtung geistiger Tätigkeit und externer Wahrnehmung ist beileibe kein Zufall, sondern veranschaulicht die Grundlage dessen, wie wir unseren Verstand erweitern. Norman [205] äußert sich dazu wie folgt:

> »The power of the unaided mind is highly overrated. Without external aids, memory, thought, and reasoning are all constrained. But human intelligence is highly flexible and adaptive, superb at inventing procedures and objects that overcome its own limits. The real powers come from devising external aids that enhance cognitive abilities. How have we increased memory, thought, and reasoning? By the invention of external aids; It is things that make us smart.«[2]

[1] »Nichts existiert im Geiste, das nicht vorher von den Sinnen wahrgenommen wurde.«

[2] »Die Kraft des nicht unterstützten Geistes wird maßlos überschätzt. Ohne externe Unterstützung sind sowohl Gedächtnis, Verstand als auch logisches Denken begrenzt. Jedoch ist der menschliche Intellekt sehr flexibel und anpassungsfähig. Er ist hervorragend darin, Vorgehensweisen und Dinge zu erfinden, die ihm helfen, die eigenen Grenzen zu überwinden.

Diese mentale Verflechtung externer und interner Repräsentationen und Prozesse zur Unterstützung der Kognition wird auch als »externe Kognition« bezeichnet [241]. Ein wichtiges Hilfsmittel dieser Art sind grafische Darstellungen.

6.1.1 Grafische Darstellungen als kognitive Hilfsmittel

Grafische Darstellungen können in zweierlei Hinsicht unterstützend wirken. Ein erster Anwendungsfall ist das Mittel zur Kommunikation, leicht veranschaulicht an dem Sprichwort: »Ein Bild sagt mehr als tausend Worte.«[3] Die Kommunikation einer Information erfordert jedoch, dass diese bereits vorliegt. Ein weiterer Anwendungsfall von grafischen Darstellungen zielt demgegenüber darauf ab, die Information selbst zu generieren oder zu entdecken. Dabei werden die besonderen Fähigkeiten der visuellen Wahrnehmung eingesetzt, um logische Probleme zu lösen. Bertin [20] formuliert dieses Vorgehen als:

> »Using vision to think.«[4]

Nach dem Nobelpreisträger Roger W. Sperry besteht das menschliche Gehirn aus zwei verarbeitenden Einheiten, jeweils lokalisiert in einer der beiden Hemisphären [260]. Dabei übernimmt die linke Hirnhälfte vorwiegend Aufgaben des verbalen, analytischen, rationalen, zeitlichen und sequentiellen Denkens, während die rechte Hirnhälfte nonverbale, synthetische, intuitive und simultane Aufgaben übernimmt. Die linke Hemisphäre befähigt damit das synchrone, quantitative Denken, wohingegen die simultanen, qualitativen Fähigkeiten der rechten Hirnhälfte ein schnelles Erfassen visueller Informationen ermöglichen. Insofern helfen grafische Darstellungen, das gesamte Potential des Verstandes auszuschöpfen, indem sie beide Hemisphären integrieren. Der Einsatz von sowohl verbalen als auch nonverbalen Repräsentationen wird häufig auch als »Dual-Coding-Theorie« bezeichnet [213].

6.1.2 Geschichte und Entwicklung

Seit jeher versuchen Menschen, wichtige Informationen bildlich zu vermitteln, indem sie beispielsweise Illustrationen, Fotografien oder technische Zeichnungen verwenden. Verdeutlicht wird die tiefe Verankerung grafischer Darstellungen in der menschlichen Kultur beispielsweise durch die frühen Tierdarstellungen in den Höhlen von Chauvet in Südfrankreich. Die dortigen Malereien entstanden in der Steinzeit, um 31500 v. Christus und gelten als die ältesten Zeugnisse vom Menschen geschaffener Abbildungen. Der Einsatz von grafischen Darstellungen in der Wissenschaft reicht zurück bis zu Euklids Abhandlung »Die Elemente«, die bereits ca. 300 v. Christus Zeichnungen einsetzt, um Eigenschaften in der Geometrie zu illustrieren und darzustellen [82]. Bereits 1637 betont René Descartes die Bedeutung der Visualisierung und insbesondere der Verwendung von Diagrammen für die wissenschaftliche Forschung [74]. Die Arbeit von Playfair [223] aus dem Jahre 1786 gehört zu den ersten, die zur Veranschaulichung von Daten abstrakte visuelle Elemente einsetzte. Grundlegend für

Seine tatsächliche Kraft entsteht aus dem Entwickeln externer Hilfsmittel, die seine kognitiven Fähigkeiten erweitern. Wie haben wir das Gedächtnis, den Verstand und das logische Denken verbessert? Durch die Erfindung externer Hilfsmittel: Es sind die Dinge, die uns schlau machen.«

[3] Nach Paul Martin Lester, Professor für Kommunikation an der Universität von Kalifornien in Fullerton, wurde dieses Sprichwort von dem Werbetexter Frederick R. Barnard im Rahmen einer Straßenbahnwerbung aufgebracht und basiert auf einem chinesischen Sprichwort [53].

[4] »Das Sehvermögen zum Denken einsetzen.«

die Entwicklung der wissenschaftlichen Visualisierung als Disziplin waren vor allem die einflussreichen Arbeiten von Bertin [20, 21] und Tufte [284]. Mit steigender Leistungsfähigkeit von Computern wurden diese zunehmend dazu eingesetzt, grafische Darstellungen zu erzeugen. Ein frühes Beispiel ist das 1958 für die Luftfahrtüberwachung entwickelte Projekt SAGE [231]. Mit dem Bericht der NSF [188] aus dem Jahr 1987 rückt die Thematik in den Blickpunkt des Interesses. McCormick und De Fanti prägen in dem Bericht den Begriff *Visualization in Scientific Computing*[5], der heute allgemein als *Scientific Visualization*[6] abgekürzt wird [38, 188, 202, 313]. Damit gilt 1987 als das Geburtsjahr der heutigen wissenschaftlichen Visualisierung im Sinne einer Disziplin [53]. Sie behandelt den Einsatz von Computergrafik zur visuellen Repräsentation wissenschaftlicher Daten, um das Verständnis zu fördern. Als Leitspruch der wissenschaftlichen Visualisierung hat folgendes Zitat von Richard Hamming besondere Bekanntheit erlangt:

> »The purpose of computing is insight, not numbers.«[7]

Die wissenschaftliche Visualisierung wird vornehmlich auf Daten aus dem wissenschaftlichen Umfeld angewendet, die einen physikalischen Hintergrund besitzen. Handelt es sich demgegenüber allgemein um abstrakte Daten ohne physikalischen Bezug, wie beispielsweise aus dem Bereich der Wirtschaft, so kommen hingegen auch Methoden aus dem Bereich der Informationsvisualisierung (*Information Visualization* [235]) zum Einsatz [53]. Heutzutage ist die Visualisierung als unersetzliches Hilfsmittel zur effizienten und effektiven Analyse großer komplexer Datenmengen aus vielen Bereichen der Wissenschaft und des Ingenieurwesens nicht mehr wegzudenken [68, 172, 197].

6.1.3 Menschlicher Analyseprozess

Während viele Bereiche der Informationstechnologie darauf abzielen, den Menschen durch Automatisierung abzulösen, sind Visualisierungssysteme hingegen explizit zu dem Zweck entworfen worden, den Menschen nicht zu ersetzen, sondern seine Fähigkeiten zu erweitern [197]. Abbildung 6.1 zeigt eine schematische Darstellung des visuellen Analyseprozesses, mit dessen Hilfe Rohdaten in Wissen transformiert werden. Der Ansatz zeichnet sich durch die Kombination der menschlichen kognitiven Fähigkeiten mit Computerkomponenten und interaktiven Benutzerschnittstellen aus. Im Mittelpunkt steht dabei der Mensch, dessen hohe visuelle Aufnahmefähigkeit sich die Visualisierung zu Nutze macht und es ihm auf diese Weise ermöglicht, Informationen um Größenordungen schneller und effizienter zu erfassen, als es beispielsweise durch direktes Analysieren der Rohdaten möglich wäre [197]. Die Visualisierung präzisiert und reduziert den Datenstrom mit Hilfe der Abstraktion (Schematisierung) und des Verwerfens überflüssiger Informationen auf schnelle und effiziente Weise [53, 197]. Durch diesen Prozess werden signifikante Komponenten betont, komplexe Eigenschaften vereinfacht und bislang verdeckte Merkmale kommen möglicherweise zum Vorschein [79]. Eine besondere Rolle nimmt dabei die hoch entwickelte Fähigkeit des Menschen zur Mustererkennung ein. Diese hilft ihm, Merkmale, Beziehungen und Anomalien in Daten zu erkennen. Im Vergleich zu Computern ist er speziell darin überlegen, unvorhergesehene und unerwartete Merkmale zu entdecken [68]. Auf diese Weise werden Wissenschaftler und Ingenieure in die Lage versetzt, Zusammenhänge aufzudecken, die andernfalls unentdeckt blieben. In Kombination mit dem a-priori-Wissen bezüglich

[5] Visualisierung im wissenschaftlichen Rechnen
[6] Wissenschaftliche Visualisierung
[7] »Das Ziel des Rechnens sind nicht Zahlen, sondern Einsicht.«

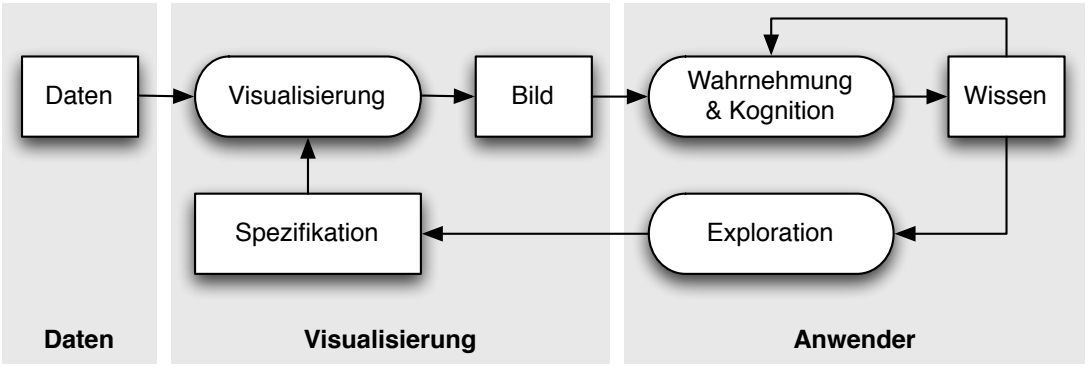


Abbildung 6.1: *Schematische Darstellung des Analyseprozesses mit Hilfe von Visualisierung nach [197] und [306].*

des betrachteten Phänomens ermöglicht die Visualisierung dem Anwender somit die Bildung eines mentalen Modells der zugrundeliegenden Daten und Informationen sowie der mit ihnen verbundenen Modelle und Prozesse [17, 197, 247, 306].

6.1.4 Ziele und Aufgaben

Ziel der Visualisierung ist eine möglichst genaue und informative visuelle Repräsentation der zugrundeliegenden Daten, so dass die relevanten Eigenschaften intuitiv erfasst, interpretiert und bewertet werden können [247]. Sie stellt damit für den Anwender ein Hilfsmittel dar, um ein Verständnis und eine Intuition für die Problemstellung zu entwickeln, auf deren Grundlage Fragen, Hypothesen und Modelle abgeleitet werden können [197, 306]. Dabei wird die Visualisierung nach Schumann [247] und Doleisch [78] auf den drei Stufen Exploration, Analyse und Präsentation eingesetzt.

Exploration

In der Phase der Exploration liegt noch keine Hypothese über die Struktur und Eigenschaften der Rohdaten vor. Den Ausgangspunkt für die Darstellung bilden die Daten selbst. Davon ausgehend erfolgt eine interaktive ungerichtete Suche nach Strukturen, Merkmalen und internen Zusammenhänge, die hinreichende Informationen über den Hintergrund der Daten liefert und damit die Formulierung einer Hypothese erlaubt. Die Visualisierungsumgebung sollte dazu ein Maximum an Flexibilität und Interaktivität bieten.

Analyse

Liegt eine Hypothese vor, werden die Strukturen und Merkmale in der anschließenden Analysephase eingehend untersucht. Die Analyse dient zur Bestätigung oder Falsifizierung der Hypothese und zur Ableitung neuer Fragestellungen. Der Interaktivität kommt auch in der Analysephase eine Schlüsselfunktion zu.

Präsentation

Den letzten Schritt stellt die Präsentation und Kommunikation der gewonnenen Erkenntnisse dar. Die Fakten sollten dabei deutlich erkennbar sein, so dass Dritte diese intuitiv erfassen können. Demgegenüber tritt die Interaktivität in den Hintergrund.

6.1.5 Qualitätskriterien

Eine Visualisierung sollte im Allgemeinen in erster Hinsicht expressiv, effektiv und zusätzlich nach Möglichkeit angemessen sowie effizient sein [53, 141, 247, 306]. Dabei ist die Expressivität [175] oder auch Ausdrucksfähigkeit eine Grundvoraussetzung einer jeden Visualisierung und erfordert, dass die zugrundeliegenden Daten möglichst unverfälscht abgebildet werden. Demzufolge dürfen nur Informationen wiedergegeben werden, die auch tatsächlich in den Daten enthaltenen sind. Dieses Kriterium steht daher in enger Relation zu dem von Tufte [284] eingeführten *Lie Factor*[8], der das Verhältnis zwischen der Größe eines in der Abbildung dargestellten Effekts und der Größe des Effekts in den Daten definiert. Die Expressivität einer Visualisierung ist primär abhängig von der Struktur und Art der Ursprungsdaten [247]. Inkorrekte Annahmen über Merkmale oder Herkunft führen zusammen mit der Komplexität der visuellen menschlichen Wahrnehmung leicht zu Repräsentationen, die ebenso irreführend wie unterstützend sein können [313].

Für eine bestimmte Datenmenge kann es durchaus mehrere Visualisierungsformen geben, die das Expressivitätskriterium erfüllen. In diesem Fall gibt die jeweilige Effektivität weiteren Aufschluss über die Qualität einer Darstellung. Sie gibt an, ob eine Darstellung intuitiv wahrgenommen werden kann. Anders als die Expressivität hängt die Effektivität einer Visualisierung jedoch nicht nur von den zugrundeliegenden Daten, sondern auch entscheidend von weiteren, zum großen Teil stark subjektiven Einflussfaktoren ab. In erster Linie sind dies die visuellen Wahrnehmungsfähigkeiten des Betrachters sowie dessen Erfahrungen und Vorwissen (a-priori-Wissen) [247, 306]. Des Weiteren haben der jeweilige Anwendungskontext, die Zielsetzung und die charakteristischen Eigenschaften des jeweiligen Ausgabemediums Einfluss auf die Effektivität [137, 175]. Damit wird deutlich, dass eine objektive Bewertung einer Visualisierung nach ausschließlich formalen Kriterien allgemein nicht möglich ist. Stattdessen ist das Ergebnis stark abhängig von dem jeweiligen Anwender und dem schwer zu wiederholenden Verlauf eines interaktiven Visualisierungsprozesses. Ein von van Wijk vorgeschlagener Ansatz zur Beurteilung der Qualität von Visualisierungen verdeutlicht diese Aussage [306].

Neben den Forderungen nach Expressivität und Effektivität sollte eine Visualisierung insbesondere im Kontext einer interaktiven Umgebung zusätzlich angemessen bzw. effizient sein. So wird verlangt, dass die Generierung der Darstellung ein minimales Maß an Ressourcen benötigt und damit auch die entstehenden Kosten gering gehalten werden [247, 306]. Dieser Aspekt wird hinsichtlich seiner Bedeutung für die Interaktivität gesondert in Abschnitt 6.3.5 behandelt.

6.1.6 Visualisierung von Strömungen

In VIRTUALFLUIDS INTERACTIVE ist die Visualisierung von Strömungsdaten von besonderer Bedeutung. Sie stellt einen der klassischen Bereiche der wissenschaftlichen Visualisierung dar. Das allge-

[8] Lügenfaktor

meine Vorgehen der Strömungsvisualisierung besteht darin, diskrete Geometrien zu erzeugen, deren Form, Größe, Orientierung und Position die Charakteristiken des zugrundeliegenden Strömungsfeldes wiedergeben [189]. Dabei besteht eine besondere Herausforderung in der häufig immensen Größe der Datensätze und ihrer Komplexität, die in erster Linie aus der Multidimensionalität, Informationsdichte und der Zeitabhängigkeit der Ergebnisdaten resultiert [42, 47, 156]. In den Daten sind im Allgemeinen Informationen über den skalaren Druck sowie die vektorielle Geschwindigkeit für jeden Punkt des Berechnungsgitters enthalten. Zusätzlich können aus dem Datensatz Strukturen wie beispielsweise Wirbel und Schockwellen abgeleitet werden. VIRTUALFLUIDS INTERACTIVE bietet verschiedene Visualisierungsformen an, die einzeln oder in Kombination für die Analyse der Strömungsdaten angewendet werden können. Die Visualisierung verwendet dabei unter anderem Methoden der Computergrafik. Jedoch ist die tendenziell abstrakte Darstellung weniger auf fotorealistische Abbildungen ausgerichtet, als vielmehr auf eine präzise Wiedergabe der Informationen aus Ergebnisdaten. Als fundamentales Mittel zur Förderung des Verständnisses bedient sich die Visualisierung der ausgeprägten räumlichen Auffassungsgabe des Menschen [53, 174]. Für die Darstellung von Raum und Volumen werden verschiedene Techniken eingesetzt, zu denen etwa die perspektivische Projektion und das Ausblenden verdeckter Linien und Flächen (*Hidden Line and Surface Removal*) zählen [313]. Einen wichtigen Aspekt stellt dabei die menschliche Fähigkeit dar, Größen verschiedener Objekte zu vergleichen. Diese Fähigkeit spielt zusätzlich auch bei der quantitativen Einschätzung von Informationen eine große Rolle. Dabei muss nach Tufte [284, 285] stets eine entscheidende Frage beantwortet werden können: »Im Vergleich zu was?«. Auf Grund des Fehlens adäquater visueller Stimuli hinsichtlich des Maßstabes, werden absolute Größen und Distanzen häufig jedoch unterschätzt. Daher werden weitere Eigenschaften wie beispielsweise Farben eingesetzt, um dem Betrachter zusätzliche Informationen zu vermitteln [53]. Farben stellen eine bewährte Methode dar, die Darstellung um Informationen zu ergänzen, ohne diese zu überladen [313]. So kann beispielsweise eine rote Farbgebung dazu eingesetzt werden, Bereiche hoher Strömungsgeschwindigkeiten anzuzeigen, wohingegen eine blaue Färbung langsame Geschwindigkeiten repräsentiert. Ein weiterer, insbesondere für die Betrachtung von zeitabhängigen Strömungsdaten, wichtiger Aspekt ist das Mittel der Animation. Neben den drei Raumrichtungen stellt die Zeit die einzige zusätzliche Dimension dar, der wir in der realen Welt begegnen. Es liegt daher nahe, die Zeitkomponente in Form einer Animation abzubilden. Die Mittel Raum, Farbe und Animation kommen in einer Vielzahl verschiedener Visualisierungstechniken auf unterschiedliche Art zur Anwendung. Eine Auswahl derer, die sich für die Auswertung von Strömungsdaten bewährt haben, steht in VIRTUALFLUIDS INTERACTIVE zur Verfügung.

Allgemein können Visualisierungsmethoden grob in lokale und globale Verfahren kategorisiert werden [131]. Globale Verfahren ermöglichen es, die Entwicklungen des gesamten Strömungsphänomens zu beobachten und unterstützen die Navigation innerhalb des Stömungsgebiets. Demgegenüber ist eine lokale Perspektive notwendig, um detaillierte Informationen entnehmen zu können. VIRTUALFLUIDS INTERACTIVE unterstützt daher sowohl globale als auch lokale Methoden.

Planare Schnitte mit Farbkodierung

Bei diesem Verfahren zur Darstellung von Skalarwerten wird ein planares Teilgebiet aus den Ergebnisdaten extrahiert. Mit Hilfe einer Transferfunktion, die jeder skalaren Feldgröße einen Farbwert zuordnet, werden die Ergebnisdaten anschließend auf der Schnittebene in Form eines Farbverlaufs

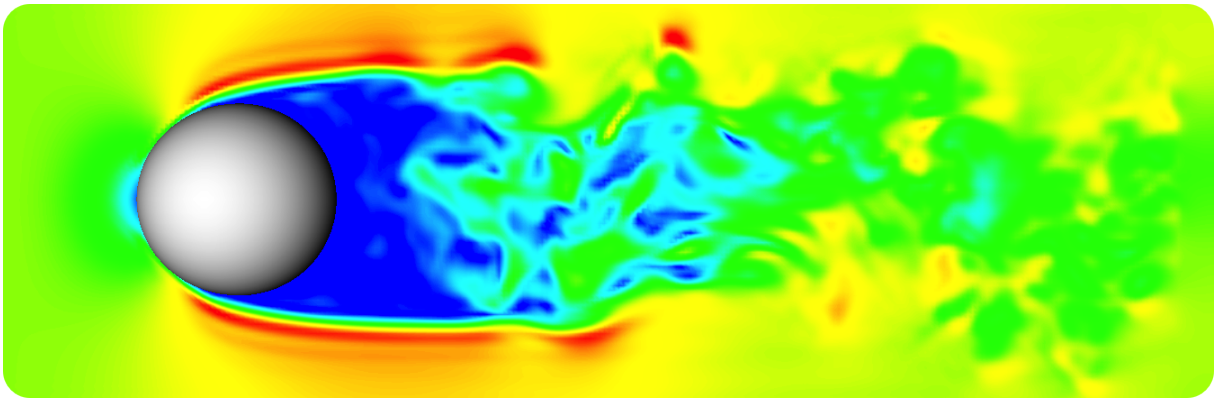


Abbildung 6.2: *Planarer Schnitt mit Farbgebung einer Kanalströmung im Umfeld einer Kugel. Farblich dargestellt ist der Betrag der Geschwindigkeit.*

dargestellt. Die Farbkodierung (*Color Mapping*) ist eine gängige Methode zur Visualisierung von Skalarwerten, die sich gut mit anderen Visualisierungstechniken kombinieren lässt. Die Methode gehört zu den direkten Visualisierungsverfahren und ist sehr effizient, da sie unter anderem einen geringen Rechenaufwand erfordert und verhältnismäßig einfach zu implementieren ist. Planare Schnitte mit Farbkodierung bieten zudem den Vorteil, dem Anwender keinerlei räumliches Vorstellungsvermögen abzuverlangen. Auf Grund der geringen visuellen Komplexität weisen planare Schnitte jedoch eine verminderte Kohärenz auf und führen zudem häufig zu Verdeckung anderer Elemente [189].

Glyphen

Ebenfalls zu den direkten Visualisierungsverfahren gehört die Darstellung mit Hilfe von Glyphen, auch Ikonen genannt. Ein prominentes Beispiel sind die sogenannten Chernoff Gesichter, die multivariante Daten in Form menschlicher Gesichter repräsentieren [57]. Allgemein sind Glyphen visuelle Elemente, die sich aus einer Reihe von Komponenten zusammensetzen, auf die jeweils verschiedene Merkmale eines Datensatzes abgebildet werden können. Glyphen eignen sich zur Darstellung von Vektordaten. Dabei werden die Glyphen in Richtung der Verktorkomponenten orientiert und zusätzlich anhand einer skalaren Größe wie beispielsweise dem Betrags des Geschwindigkeitsvektors skaliert oder auch farbkodiert. Dafür eignen sich beispielsweise Pfeile oder auch Kegel. Die Glyphen-Technik sollte jedoch mit Bedacht angewendet werden. Häufig ist es bei der Darstellung von 3D-Daten problematisch, Position und Orientierung der Glyphen korrekt zu erkennen. Zudem kann eine große Zahl an Glyphen zusätzlich zu Verwirrung führen. Vorsicht ist insbesondere bei der Skalierung von Glyphen in Hinblick auf Tuftes »Lügenfaktor« (vgl. Abschnitt 6.1.5) angebracht, da eine Skalierung eines 3D-Glyphen eine nichtlineare Änderung des Erscheinungsbildes zur Folge hat. Stattdessen nimmt die Oberfläche eines Glyphen mit dem Quadrat des Skalierungsfaktors zu. Da dieser Aspekt häufig im Kontext von Skalierungen vorzufinden ist, sollte darauf geachtet werden, irreführenden Darstellungen vorzubeugen [132].

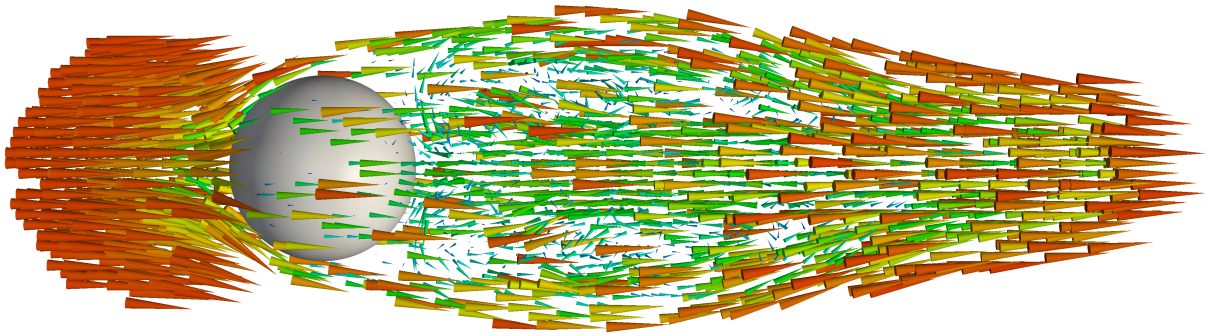


Abbildung 6.3: *Glyphen mit Farbgebung einer Kugelumströmung. Dargestellt ist das Geschwindigkeitsfeld.*

Stromlinien

Als effektives Visualisierungsverfahren, mit dessen Hilfe insbesondere Wirbelstrukturen gut identifiziert werden können, haben sich Stromlinien bewährt. Diese gehören zu der Gruppe sogenannter integrationsbasierter Verfahren. Eine Stromlinie ist formal definiert als Kurve, die in jedem Punkt tangential zu dem Geschwindigkeitsvektor eines momentanen Strömungsfeldes ist und durch einen festgelegten Raumpunkt, auch Saatpunkt genannt, verläuft [155]. Sogenannte Tufts sind sehr kurze Stromlinien, die durch das Darstellen des eigentlichen Geschwindigkeitsvektors angenähert werden können (vgl. Glyphen). Zumeist werden Stromlinien jedoch mit Hilfe des numerischen Integrationsverfahrens nach Runge-Kutta mit einer Genauigkeit zweiter Ordnung berechnet. Eine Stromlinie ist unphysikalisch in dem Sinne, dass sie auf einem momentanen Zeitpunkt basiert und kein Element des Fluides tatsächlich der Trajektorie folgt[9]; dennoch geben Stromlinien einen sehr guten Überblick über die globale instantane Struktur des Geschwindigkeitsfeldes. Besonderer Beachtung bedarf die Anzahl der Stromlinien. Sie muss mit Bedacht gewählt werden, da zu viele Stromlinien leicht zu einer unübersichtlichen hohen visuellen Komplexität und Verdeckung führen. Demgegenüber kommen bei einer zu geringen Anzahl an Stromlinien wichtige Charakteristiken eventuell nicht zum Vorschein. Die Anzahl der Stromlinien beeinflusst weiterhin den Rechenaufwand des Verfahrens in erheblichem Maße. Im Allgemeinen ist das numerische Integrationsverfahren aufwändig und lediglich hinsichtlich der Anzahl zu parallelisieren, da es inhärent iterativ ist, wobei jeder Integrationsschritt auf dem vorherigen Ergebnis basiert. Dieser Aspekt ist speziell in einer interaktiven Umgebung von besonderer Relevanz. Der räumliche Eindruck von Stromlinien kann mit Hilfe sogenannter Stromröhren zusätzlich gesteigert werden. Dabei wird eine Kreislinie an jedem Integrationspunkt normal zur Stromlinie platziert. Die anschließend miteinander verbundenen Kreislinien formen eine geschlossen Röhre. Mittels des Durchmessers der Stromröhre kann zusätzliche die lokale Geschwindigkeit des Strömungsfeldes repräsentiert werden.

[9] Der tatsächliche Pfad eines Elemtes im zeitlich veränderlichen Stömungsfeld wird auch als Bahnlinie bezeichnet. Im Fall einer stationären Strömung sind Strom- und Bahnlinien identisch.

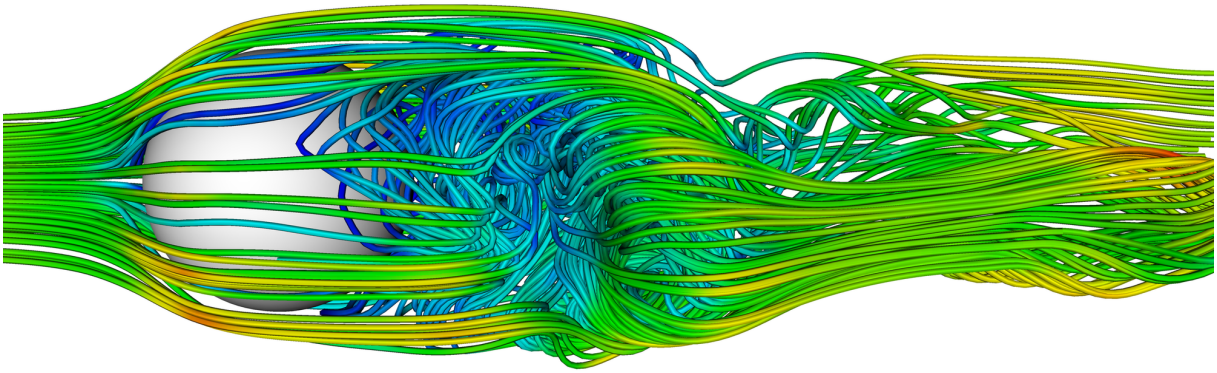


Abbildung 6.4: *Stromlinien mit Farbgebung einer Kugelumströmung. Dargestellt ist das Geschwindigkeitsfeld, das dem Betrag der Geschwindigkeit nach eingefärbt ist.*

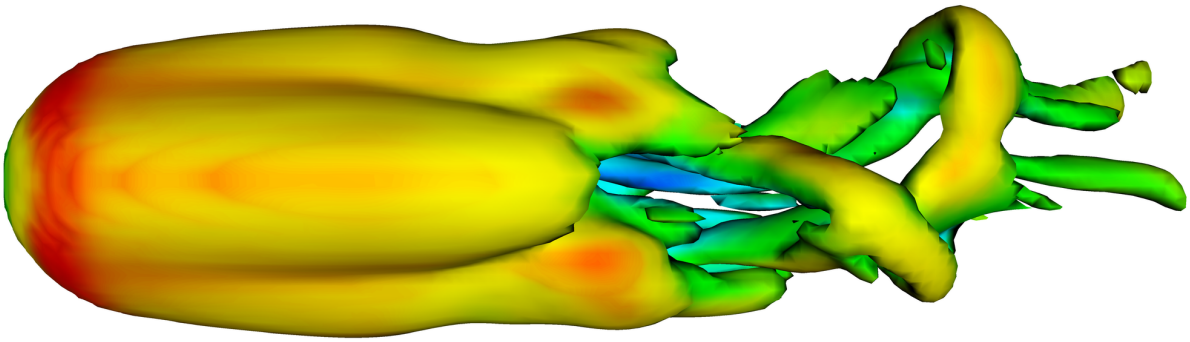


Abbildung 6.5: *Isofläche mit Farbgebung der Umströmung einer Kugel. Dargestellt ist die Isofläche für eine bestimmte Wirbelstärke, die zusätzlich nach dem Betrag der Geschwindigkeit eingefärbt ist.*

Isoflächen

Ein direktes Verfahren zur Volumenvisualisierung stellen Isoflächen dar, mit deren Hilfe skalare dreidimensionale Datensätze in Form von Flächen dargestellt werden können. Diese eignen sich insbesondere dafür, die Entwicklung einer Strömung zu beobachten und transiente Merkmale zu identifizieren. Für die Berechnung der Isoflächen für einen bestimmten skalaren Schwellenwert wird in aller Regel der von Lorensen und Cline vorgestellte *Marching-Cubes*-Algorithmus eingesetzt [169]. Dieser erfordert einen hohen Rechenaufwand, kann jedoch effizient parallelisiert werden.

6.2 Visualisierungsprozess

Das Ziel der Visualisierung besteht darin, beliebige Ergebnisdaten so umzuwandeln, dass sie von der menschlichen Wahrnehmung erfasst werden können und dabei die Integrität der Informationen gewährleistet ist. Dieser Prozess kann als eine Serie sukzessiver Transformationen angesehen werden, mit deren Hilfe die unter Umständen abstrakten, nicht dreidimensionalen geometrischen Ergebnisdaten in zweidimensionale Bilder überführt werden [240, 247, 313].

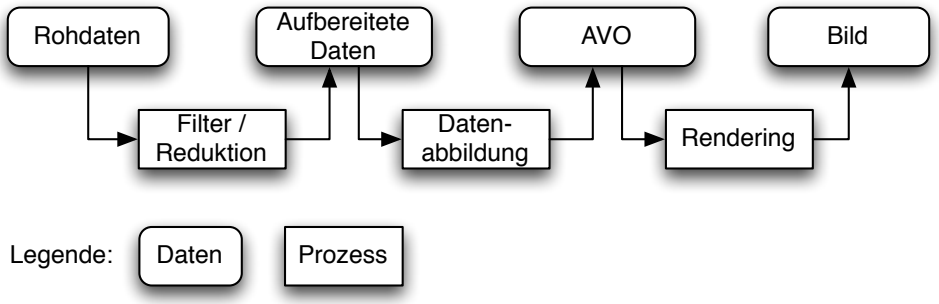


Abbildung 6.6: *Datenfluss-Visualisierungspipeline nach Haber und McNabb [112].*

6.2.1 Visualisierungspipeline

Ein in der wissenschaftlichen Visualisierung sehr populäres Referenzmodell ist das von Haber und McNabb vorgestellte Datenflussmodell [111]. Die einzelnen Visualisierungsschritte formen dabei einen Graphen, schematisch dargestellt in Abbildung 6.6. In diese sogenannte Visualisierungspipeline fließen Rohdaten auf der einen Seite hinein und münden auf der anderen Seite in Bildern. Der Visualisierungsprozess untergliedert sich dabei auf übergeordneter Ebene in zwei wesentliche Aspekte: Transformationen und (generierte) Daten. Zu den Transformationen gehören die Datenaufbereitung (*Filtering*), Datenabbildung (*Mapping*) und das abschließende Zeichnen der Darstellung (*Rendering*). Analog zu den Transformationsschritten wird zwischen folgenden Daten unterschieden: Rohdaten, aufbereitete Daten (*Derived Data*), Geometriedaten oder auch abstraktes Visualisierungsobjekt (AVO) (*Abstract Visualization Object*) und schließlich Bilddaten. Die Pipeline beginnt mit dem *Filtering* der Rohdaten. Im ursprünglichen Sinne deutet der Begriff auf ein selektives Entfernen bestimmter Daten hin. An dieser Stelle wird die Bezeichnung jedoch in ihrem ingenieurmäßigen Sinne verstanden und bedeutet lediglich, Daten von einer Form in eine andere zu transformieren. Die häufig alternativ verwendeten Bezeichnungen *»Data Enrichment«* und *»Data Enhancement«* werden demgegenüber deutlich konkreter. Ein Aufbereiten oder Anreichern der Daten erfolgt häufig in Form von Interpolation, Rauschunterdrückung oder auch Schwellenwertberechnung, um beispielsweise Fehler zu korrigieren, Daten zu glätten oder Werte nach bestimmten Kriterien zu extrahieren und dabei die Datenmenge zu reduzieren. Bei der Anwendung muss jedoch auf die Einführung von Effekten wie dem *Aliasing*[10] geachtet werden.

Die aufbereiteten Daten bilden die Eingangsdaten für das *Mapping* und somit das Kernstück des Datenflussmodells. In diesem Schritt kommen unter anderem die in Abschnitt 6.1.6 beschriebenen Visualisierungsmethoden zur Anwendung. Abhängig von dem gewählten Visualisierungsverfahren findet eine Daten-zu-Geometrie-Abbildung statt, deren Ergebnis ein abstraktes Visualisierungsobjekt darstellt. Dabei werden relevante Merkmale der Eingangsdaten auf Attribute des AVOs abgebildet. Zu den Attributen zählen beispielsweise Größe, Farbe, Transparenz und Texturen des AVOs.

[10] Als *Aliasing*-Effekt werden im Bereich der Signalanalyse Fehler bezeichnet, die durch Missachtung niedriger Frequenzen beim digitalen Abtasten von Signalen auftreten. In der Bildverarbeitung und Computergrafik treten *Aliasing*-Effekte bei der Abtastung von Bildern auf und führen zu Mustern, die im Originalbild nicht enthalten sind. [67]

Das abschließende *Rendering* überführt das abstrakte Visualisierungsobjekt in eine visuelle Darstellung. Üblicherweise wird das AVO dabei perspektivisch dargestellt. Dazu wird das Objekt möglicherweise rotiert, skaliert oder verschoben, um dem Benutzer eine effektive Repräsentation zu bieten.

Das Datenflussmodell bringt einige wertvolle Vorteile mit sich, weist aber auch gewisse Limitierungen auf, die insbesondere in interaktiven Umgebungen berücksichtigt werden müssen. Beispielsweise erlaubt das Datenflussmodell, auf einfache Art Animationen zu erstellen, indem mehrere Rohdatensätze von der selben Visualisierungspipeline verarbeitet werden. Bei einer gefüllten Pipeline ist dabei der Durchsatz unabhängig von der Anzahl der Komponenten und wird ausschließlich von der Stufe bestimmt, deren Ausführung am längsten andauert. Das erstmalige Füllen der Pipeline kann bei großen Datenmengen erhebliche Zeit in Anspruch nehmen. Bleiben demgegenüber die Rohdaten identisch und wird stattdessen eine Stufe der Visualisierungspipeline variiert, so müssen lediglich alle nachfolgenden Stufen neu ausgeführt werden, wodurch eine hohe Effizienz erreicht wird.

Ein bekanntes Problem des Datenflussmodells besteht in einem oftmals großen Speicherbedarf [59, 215, 289]. Dieser resultiert häufig daraus, dass jede Stufe eine Kopie der Zwischenergebnisse im Speicher vorhält. Dieser Umstand macht sich bei sehr großen Datenmengen negativ bemerkbar. Es ist jedoch wichtig anzumerken, dass die Einschränkungen des Datenflussmodells nicht inhärent sind, sondern im Wesentlichen von der jeweiligen Implementierung abhängen. Song und Golin haben gezeigt, dass die Limitierungen unter anderem mit Hilfe eines feingranularen Datenflussmodells reduziert werden können [259].

6.2.2 Verteilte Ausführung der Visualisierungspipeline

Die meisten Softwarepakete im Bereich der Visualisierung und des *Computational Steerings* sehen vor, die vollständige Visualisierungspipeline lokal auf einer einzelnen leistungsstarken Grafik-*Workstation* auszuführen [36]. In einer typischen HPC-Simulationsumgebung ist es daher erforderlich die rohen Ergebnisdaten für die Analyse über ein Netzwerk an das Visualisierungssystem zu übermitteln. Diese Kommunikation zwischen Hochleistungsrechner und Visualisierungssystem stellt in der Regel einen Flaschenhals dar [18, 36, 165, 315]. Generell erlaubt die Visualisierungspipeline jedoch auch eine partitionierte Ausführung der einzelnen Stufen auf mehrere Systeme. Beispiele dafür demonstrieren unter anderem Wood [310] und Brodlie [34]. Durch eine verteilte Ausführung der Visualisierungspipeline, bei der einzelne Stufen beispielsweise zusammen mit der Simulation auf einem Hochleistungsrechner ausgeführt werden, kann sowohl die Menge der zu kommunizierenden Daten als auch die Last auf das Visualisierungssystem reduziert werden. Nachteilig dabei ist jedoch, dass bei zunehmender Verteilung die Möglichkeiten zur Interaktion deutlich eingeschränkt werden [45, 154, 247]. Die Co-Lokalität von Simulations- und Visualisierungsalgorithmik wird häufig auch als In-Situ-Visualisierung bezeichnet und ist insbesondere bei sehr großen Datenmengen von Bedeutung [140, 173, 318]. In VIRTUALFLUIDS INTERACTIVE werden alle Anwendungsteile auf einem System ausgeführt. Eine limitierende Netzwerkkommunikation ist aus diesem Grunde nicht vorhanden. Stattdessen werden die Ergebnisdaten aus dem Hauptspeicher der GPU in den Hauptspeicher des Systems transferiert. Dieser Vorgang ist ebenfalls zeitaufwändig, im Praxisvergleich jedoch mehr als 60 mal schneller als eine Netzwerkkommunikation [86]. Obgleich nicht in gleichem Maße ausschlaggebend, ist eine In-Situ-Bearbeitung jedoch auch hier von Vorteil, um etwa Speicherbewegungen auf ein Minimum zu reduzieren und die Last für die Visualisierungspipeline zu senken.

6.2.3 The Visualization Toolkit (VTK)

Im Rahmen dieser Arbeit wird die frei verfügbare Bibliothek »*The Visualization Toolkit*« (VTK) eingesetzt [12, 145, 244, 245, 246]. VTK ist ein Softwarepaket für 3D-Computergrafik, Bildverarbeitung und Visualisierung, basierend auf dem traditionellen Datenflussmodell von Haber und McNabb. Der Entwurf der *Open Source* C++ Bibliothek wurde erheblich von den Prinzipien der Objektorientierung beeinflusst. Die Verfügbarkeit des Quellcodes ermöglicht es, die Funktionalität zu erweitern und anzupassen. Zudem unterstützt VTK alle gängigen Anwendungsplattformen, wodurch Entwicklung und Portabilität deutlich erleichtert werden.

Die VTK-Nomenklatur bezeichnet verarbeitende Stufen der Visualisierungspipeline als Prozessobjekte (*Process Objects*), während Eingangs- und Zwischenergebnisdaten Datenobjekte (*Data Objects*) genannt werden. Initiiert wird die VTK-Pipeline mit sogenannten Quellobjekten (*Sources*). Diese liefern die Rohdaten für die Datenaufbereitung (*Filter*) und die anschließende Datenabbildung (*Mapper*). Aktoren (*Actors*) repräsentieren die abstrakten Visualisierungsobjekte, die schließlich von einem *Renderer* in das Anwendungsfenster (*Render Window*) gezeichnet werden. VTK ermöglicht die Benutzerinteraktion in dem Darstellungsfenster mit Hilfe verschiedener Eingabegeräte wie beispielsweise einer Maus über sogenannte *Interactor*-Objekte.

6.3 Monitoring - Erweiterte Visualisierungspipeline

Strömungssimulationen auf Basis des Lattice-Boltzmann-Verfahrens entwickeln sich auch bei relevanten Gitterauflösungen hinreichend schnell, um ein zweckmäßiges *Computational Steering* zu ermöglichen. Das gilt gleichermaßen für traditionelle Hochleistungssysteme als auch für moderne Many-Core-Architekturen (vgl. Abschnitt 5.2). Ein wesentlicher Bestandteil einer solchen interaktiven Simulation ist das *Monitoring* laufender Berechnungen. Dieses stellt eine Erweiterung der klassischen Visualisierungspipeline dar, bei der kontinuierlich generierte Ergebnisdaten direkt in die Pipeline eingeleitet werden (vgl. Abbildung 2.9). Die Leistungsfähigkeit eines solchen Systems wird dabei im Wesentlichen von der Ausführungsgeschwindigkeit der beteiligten Komponenten und der Größe und Komplexität der Ergebnisdaten bestimmt.

6.3.1 Aktualisierungszeit

Die Leistungsfähigkeit bestimmt die in einer interaktiven Umgebung kritische Aktualisierungszeit T_A (*end-to-end lag time* [274]). Diese bezieht sich auf die Zeitspanne, die für die Ausführung aller beteiligten Prozesse von der Simulation bis zur Darstellung der Ergebnisse auf dem Bildschirm benötigt wird. Im Sinne einer hohen Anwendungsqualität und Benutzerfreundlichkeit stellt die Minimierung dieser Aktualisierungszeit ein übergeordnetes Ziel dar. Daraus resultiert in einem idealen System eine Echtzeitströmungssimulation mit flüssiger Visualisierung und hoher Responsivität, wie sie heute schon in der Computerspieleindustrie vorzufinden ist [65]. Auf diese Weise wird insbesondere die Möglichkeit zur Exploration transienter Strömungsphänomene erheblich unterstützt. Die Größe und Komplexität der Ergebnisdaten macht es jedoch sehr schwer, dieses Ziel zu erreichen. Generell ist es dafür erforderlich, einen Kompromiss aus Genauigkeit und Geschwindigkeit einzugehen [17, 44, 45].

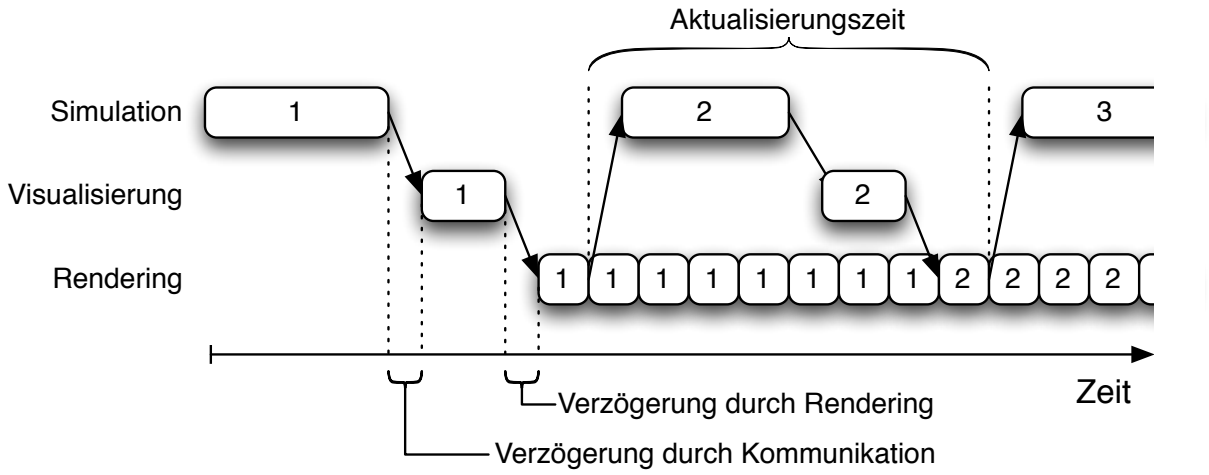


Abbildung 6.7: *Aktualisierungszeit bei sequentiell ausgeführter Visualisierungspipeline basierend auf [18].*

Aufgabe der Simulationsumgebung ist es, dem Benutzer zu ermöglichen, diesen Kompromiss selbst zu finden, anstatt eine Lösung vorzugeben [43].

Die Aktualisierungszeit wird in erheblichem Maße von der benötigten Zeit für die Visualisierung der Daten T_{vis} und der Transferdauer der Rohdaten zwischen Simulation und Visualisierungspipeline $T_{\text{sim}\rightarrow vis}$ dominiert. Weiterhin stellt die Simulationsdauer T_{sim} eine entscheidende Einflussgröße dar. Von untergeordneter Bedeutung für die Aktualisierungszeit ist hingegen die Dauer des *Renderings* T_{ren}. Im klassischen *Post-Processing*-Verfahren entspricht die Aktualisierungszeit der Summe aller Laufzeiten der sequentiell ausgeführten Prozesse (vgl. Abbildung 6.7):

$$T_A = T_{\text{sim}} + T_{\text{sim}\rightarrow\text{vis}} + T_{\text{vis}} + T_{\text{ren}} \tag{6.1}$$

Wenn jedoch wie in VIRTUALFLUIDS INTERACTIVE ein kontinuierlicher Strom an Ergebnisdaten verarbeitet wird, können die beteiligten Prozesse anteilig parallel ausgeführt werden [45]. Auf diese Weise kann die Aktualisierungszeit auf die längste Laufzeit der beteiligten Prozesse reduziert werden:

$$T_A = max(T_{\text{sim}} + T_{\text{sim}\rightarrow\text{vis}}, T_{\text{vis}}, T_{\text{ren}}) \tag{6.2}$$

Abbildung 6.8 zeigt ein Beispiel, in dem die Simulationskomponente die längste Laufzeit aufweist. Belleman spricht in diesem Fall von einem simulationsgetriebenen System [17]. In einem hochinteraktiven Anwendungsfall wird man hingegen versucht sein, die Komponenten so einzustellen, dass sich $T_{\text{sim}} + T_{\text{sim}\rightarrow\text{vis}}$ und T_{vis} annähern.

6.3.2 Zeitsynchronität

Generell ist bei einer parallelen Ausführung von Komponenten zu beachten, dass die Darstellung der Ergebnisdaten nicht mehr unmittelbar synchron mit der fortschreitenden Simulation ist. Obwohl sich dieser Aspekt im Rahmen der vorliegenden Arbeit nicht offensichtlich nachteilig ausgewirkt hat, sollte er nicht missachtet werden. Ein erprobtes Konzept zum Umgang mit dieser Problematik stellt das von Brodlie et al. vorgestellte *Checkpointing* dar [35].

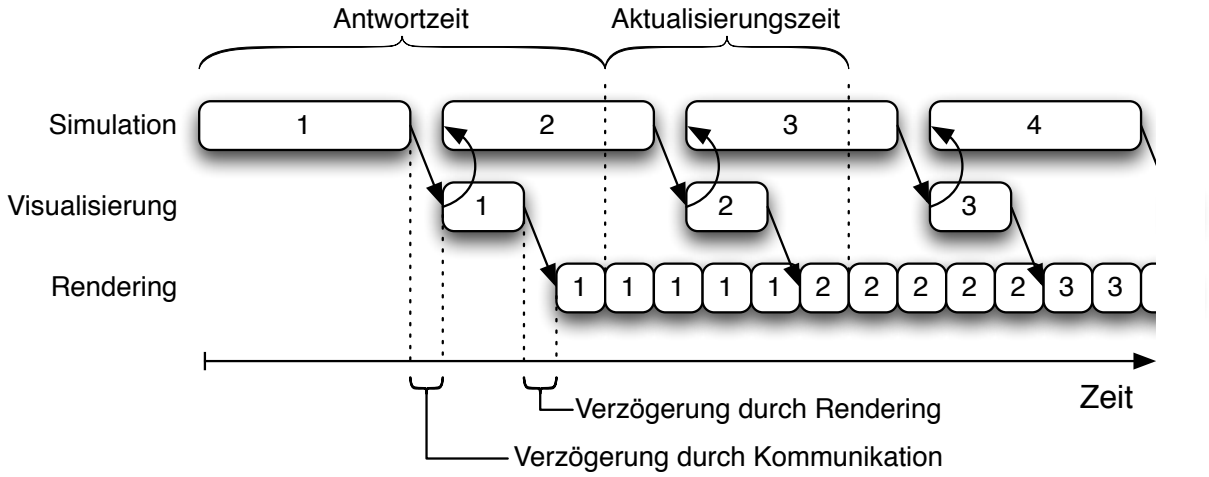


Abbildung 6.8: *Aktualisierungszeit bei asynchron ausgeführter Visualisierungspipeline basierend auf [18].*

6.3.3 Mantra der Informationssuche

Das Balancieren der Laufzeiten von Visualisierung, Simulation und Datenkommunikation stellt in einer interaktiven Umgebung eine besondere Herausforderung dar, für die keine allgemeingültige Lösung existiert. Das Vorgehen ist stattdessen stark von dem jeweiligen Anwendungsfall und den Präferenzen des Anwenders abhängig. Im Allgemeinen wird das System auf Grund der CFD-typischen großen Menge komplexer Ergebnisdaten stetig an die Leistungsgrenzen getrieben. Das betrifft Visualisierungskomponenten, die komplexe Algorithmen zur Extraktion geometrischer Primitive wie z.B. Stromlinien verwenden. Zusätzlich verstärkt wird das Problem dadurch, dass für eine informative Visualisierung komplexer Datensätze im Allgemeinen mehrere Verfahren gleichzeitig zum Einsatz kommen [21]. Daraus entsteht ein Konflikt zwischen einer möglichst genauen wissenschaftlichen Visualisierung und einer responsiven, flüssig animierten Darstellung. Der notwendige Kompromiss zwischen den konkurrierenden Eigenschaften sollte dabei nach Möglichkeit vom Anwender getroffen werden können. VIRTUALFLUIDS INTERACTIVE unterstützt den Anwender in diesem Fall mit Hilfe des von Shneiderman vorgeschlagenen »Mantra der Informationssuche« [253]:

> »Overview first, zoom and filter, then detail on demand!«[11]

Damit wird ein Verfahren beschrieben, das dem Anwender erlaubt, in einem weniger genauen aber dafür responsiven Modus einen Überblick zu gewinnen und entdeckte Phänomene bei Bedarf detailliert in einem entsprechend weniger responsiven Modus mit höherer Genauigkeit zu untersuchen. Dieses allgemein formulierte Vorgehen findet in VIRTUALFLUIDS INTERACTIVE auf mehreren Ebenen Anwendung.

6.3.4 Kommunikation zeitlich ausgedünnter Ergebnisdaten

Eine Möglichkeit, die Last auf die Visualisierungspipeline zu reduzieren, besteht darin, den Strom der Ergebnisdaten zeitlich auszudünnen. Dabei werden die Ergebnisdaten nach einer definierten Anzahl

[11] »Verschaffe dir erst einen Überblick, zoome und filtere. Bei Bedarf lass dir im Anschluss Details anzeigen«

an Zeitschritten (Aktualisierungsintervall) im sogenannten *Push*-Verfahren an die Pipeline übergeben. Das variable Aktualisierungsintervall bietet dem Benutzer dabei die Möglichkeit, die Dauer der Simulationsphase T_{sim} sehr fein zu justieren und an die jeweiligen Rahmenbedingungen anzupassen. Ein Ausdünnen (*Subsampling*) der Ergebnisdaten in dieser Form hat zumeist keinen Einfluss auf die Genauigkeit der Visualisierung [43]. Jedoch reduziert sich die Darstellung bei einem großem Aktualisierungsintervall auf eine lose, unkohärente Abfolge von Momentaufnahmen, so dass individuelle Strömungsphänomene in einzelnen Zeitschritten übersehen werden können. Das übergeordnete Ziel sollte also sein, das Aktualisierungsintervall nach Möglichkeit gering zu halten, um eine flüssig animierte Visualisierung zu erhalten.

Zu der jeweiligen Laufzeit eines Simulationsintervalls T_{sim} addiert sich die Zeit für die Kommunikation der Daten an die Visualisierungskomponente $T_{\text{sim}\rightarrow\text{vis}}$. In traditionellen Simulationsumgebungen erfordert diese Übertragung den Einsatz von Netzwerkkommunikation und stellt damit einen erheblichen Flaschenhals dar (vgl. Abschnitt 5.4). Im direkten Vergleich dazu verläuft der Transfer der Daten in VIRTUALFLUIDS INTERACTIVE auf Grund hoher Datenlokalität und Bandbreite sowie geringer Latenz mehr als 60-fach effizienter. Zudem erfolgt vor der Übermittlung eine Reduktion des Datenvolumens durch semantisches Filtern [68]. Dabei werden die makroskopischen Größen aus den LB-Verteilungen ermittelt, was einer verlustbehafteten Kompression gleichkommt. Jedoch werden Anwender überwiegend an makroskopischen Größen und weniger an den einzelnen Verteilungen interessiert sein.

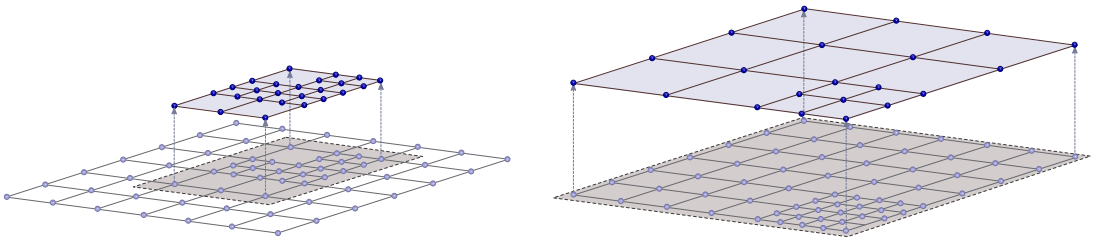
6.3.5 Zeitkritische Visualisierung

Im Gegensatz zu Anwendungen, die darauf abzielen, die reale Welt mit Hilfe aufwändiger fotorealistischer Verfahren abzubilden, ist die Darstellung wissenschaftlicher Visualisierungen zu einem gewissen Grad beliebig [43, 44]. Demnach kann die Visualisierungsform so gewählt werden, dass die äußeren Leistungsanforderungen nach Möglichkeit erfüllt werden. Bryson erhebt dieses Vorgehen zu einem allgemeinen Prinzip der Entwicklung von interaktiven Visualisierungssystemen [44]:

> »Keep the graphical rendering as simple as possible!«[12]

Damit wird gefordert, Informationen generell nur mit der Genauigkeit und Qualität darzustellen, die zum Lösen einer Aufgabe erforderlich ist. Vor dem Hintergrund, dass die Expressivität eines Visualisierungverfahrens schwer zu bestimmen und noch weniger vorauszusagen ist (vgl. Abschnitt 6.1.5), können verschiedene Verfahren in dieser Hinsicht zunächst als gleichwertig angesehen werden. Für die Wahl der Darstellungsart sollte demnach primär die Effizienz bzw. Angemessenheit des Verfahrens ausschlaggebend sein. Damit sollten weniger aufwändige Verfahren wie beispielsweise planare Schnitte oder Glyphen bevorzugt werden. Kommen hingegen rechenintensive Verfahren zum Einsatz, so kann die Last auf die Visualisierungspipeline und damit die Laufzeit T_{vis} mit Hilfe einfacher Datenreduktionsverfahren gesenkt werden. Dabei wird nach dem »Mantra der Informationssuche« verfahren (vgl. Abschnitt 6.3.3). Der Anwender hat dazu erstens die Möglichkeit, die Visualisierung auf einen bestimmten Ausschnitt des Ergebnisraumes zu begrenzen (*Cropping*) und die Datenmenge zweitens auszudünnen (*Subsampling*). Beide Reduktionsverfahren sind als zusätzliche Filterstufen umgesetzt, die sowohl einzeln als auch in Kombination angewendet werden können (vgl. Abbil-

[12] »Halte die grafische Darstellung so einfach wie möglich!«



(a) *Reduktion auf einen Teilausschnitt von 3x4 Gittergröße.*

(b) *Ausdünnen auf eine 4x4 Gittergröße.*

Abbildung 6.9: *Datenreduktion im zweidimensionalen Raum bei einer 7x7 Gittergröße.*

dung 6.9). Dabei ist hervorzuheben, dass beide Implementierungen direkt auf den Rohdaten arbeiten und somit kein zusätzlicher Speicherbedarf in Folge redundanter Datenhaltung entsteht. Eine frühe Demonstration der Reduktion auf ein Teilvolumen im Kontext der Visualisierung von Strömungsdaten erfolgte beispielsweise von Gerald-Yamasaki et al. [46] und ist heute ein gängiges Verfahren mit geringem Einfluss auf die Qualität der Darstellung. Das örtliche Ausdünnen von Ergebnisdaten wurde erstmals von Smith [257] vorgeschlagen und entspricht einer speziellen Form des von Globus [107] vorgestellten *Extract*-Konzepts. Bei diesem Vorgehen arbeiten die Visualisierungsverfahren nicht auf jedem vorhandenen Datenpunkt, sondern beispielsweise nur auf jedem zweiten. Diese Form der Datenreduktion erfordert jedoch besondere Aufmerksamkeit seitens des Anwenders, da sie leicht zu Effekten wie beispielsweise *Aliasing* führt und das Ergebnis verfälschen kann [45, 306]. Bereits ein Ausdünnen mit einem Faktor von zwei kann dabei zu qualitativ inkorrekten Ergebnissen führen [43]. Dennoch liefert auch diese Form der Visualisierung Aufschluss über das globale Strömungsverhalten und lenkt den Anwender auf Regionen, die als Teilvolumen bei voller Auflösung detailliert betrachtet werden können.

6.4 Intuitive interaktive Exploration

Der Schlüssel zu einer erfolgreichen *Computational Steering*-Umgebung sind die Interaktionsfähigkeiten [44]. Diese wirken in vielfacher Hinsicht unterstützend auf den Explorationsprozess [55, 172, 305]. Eine besondere Herausforderung für intuitive Interaktionsmöglichkeiten besteht dabei in der inhärenten Unidirektionalität des traditionellen Referenzmodells von Haber und McNabb, das die Visualisierung ursprünglich als logisches Ausgabemedium versteht [314]. Grundsätzlich erlaubt es die Pipeline jedoch, dass der Anwender einzelne Stufen zur Laufzeit modifiziert (vgl. Abbildung 6.10).

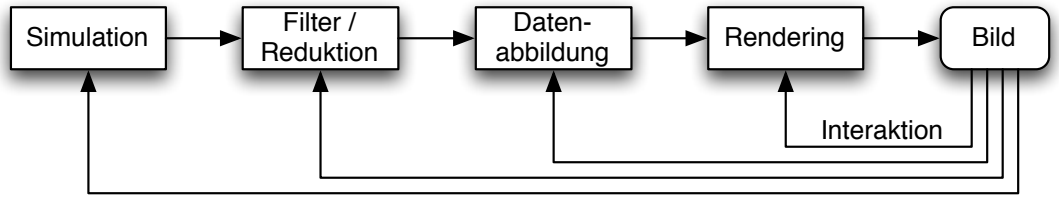


Abbildung 6.10: *Benutzerinteraktion innerhalb von* VIRTUALFLUIDS INTERACTIVE.

6.4.1 Echtzeitanimation der Szene

Ein wesentliches Merkmal der Interaktivität besteht darin, die Visualisierung in Echtzeit zu animieren. Der Anwender hat dabei die Möglichkeit, sich innerhalb der virtuellen Szene zu bewegen, indem er die Ansicht rotiert, translatiert oder skaliert. Auf diese Weise wird der räumliche Eindruck der Darstellung unterstützt. Die veränderte Perspektive erfordert im Allgemeinen eine Aktualisierung der Pipeline. In der Regel bleibt dabei das abstrakte Visualisierungsobjekt unverändert, so dass lediglich die *Rendering*-Stufe periodisch ausgeführt wird, während die übrigen Stufen keiner Aktualisierung bedürfen. Auf Grund der Trägheit des Auges, der sogenannten Nachbildwirkung [62], sollten dabei zwischen 14 bis 16 Bilder pro Sekunde erzeugt werden, um den Eindruck einer flüssig animierten Interaktionsbewegung zu erzielen[43, 152]. Eine höhere Bildrate trägt allerdings erheblich zu gesteigertem Anwendererlebnis bei, wohingegen ein Unterschreiten des Grenzwertes die effiziente Benutzerinteraktion behindert [146]. Es wird daher angestrebt, die größtmögliche Bildrate zu erzielen. Dadurch wird die Forderung unterstrichen, die grafische Darstellung so einfach wie möglich zu halten, da eine höhere Komplexität mit einer gesteigerten Laufzeit des *Renderings* einhergeht. Heute verfügbare *Workstations* erfüllen die grafischen Anforderungen von VIRTUALFLUIDS INTERACTIVE jedoch mit relativer Leichtigkeit, da typische Szenen oft aus nicht mehr als einer Million Dreiecken bestehen. Wird jedoch eine zu geringe Bildrate erzielt, so wird die Detailstufe automatisch reduziert, um den Grenzwert einzuhalten [96]. Dabei werden komplexe Strukturen beispielsweise auf primitive Geometrien temporär reduziert, um den Aufwand für die Darstellung zu senken (*LOD-Rendering*).

6.4.2 Direkte Manipulation

Neben den Leistungsanforderungen an das System stellt eine nach Möglichkeit intuitive und effiziente Benutzerinteraktion eine wichtige Grundvoraussetzung für eine effektive Exploration dar [44]. Die Interaktionsmechanismen sollten demnach so natürlich wie möglich gestaltet sein, eine unzweideutige Bedienung ermöglichen und dem Anwender geringstmögliche Aufmerksamkeit abverlangen [204]. In VIRTUALFLUIDS INTERACTIVE werden diese Forderungen mit Hilfe des von Shneiderman vorgeschlagenen Konzepts der direkten Manipulation umgesetzt [252, 254]. Im Kontext der Visualisierung ermöglicht es dem Anwender beispielsweise, analog zu dem Vorgehen in der Realität, virtuelle Objekte mit Hilfe der Computermaus zu ergreifen und zu modifizieren. Anders als im klassischen unidirektionalen Datenflussmodell stellt die grafische Darstellung somit sowohl ein logisches Ausgabe- als auch Eingabemedium dar. Den Gegensatz dazu bildet die indirekte Manipulation. Hier erfolgt die Modifikation der virtuellen Objekte durch Kopplung bestimmter Attribute an konventionelle Steuerelemente wie Schaltflächen oder Schieberegler. Diese Form der Manipulation besitzt den Vorteil, dass Parameteränderungen sehr präzise durch Eingabe des gewünschten Wertes in beispielsweise ein Textfeld erfolgen können. Ein großer Nachteil besteht jedoch darin, dass für diese Form der Eingabe ein störender Kontextwechsel von der virtuellen dreidimensionalen Darstellung zu den zweidimensionalen Bedienelementen erforderlich ist. Die direkte Manipulation ist hingegen meist unpräziser, jedoch deutlich intuitiver und kann somit in vielerlei Hinsicht vorteilhaft für den Explorationsprozess sein. So erlaubt sie beispielsweise eine deutlich effektivere Erkundung des Datenraumes auf Grund der möglichen räumlichen Orientierung. Zur Veranschaulichung dieser Aussage kann ein Szenario herangezogen werden, in dem ein Anwender ein turbulentes Geschwindigkeitsfeld unter Einsatz von Stromlinien analysieren möchte. Diese erfordern die Vorgabe eines Saatpunktes als Ausgangspunkt

für die Integration. Diesen Ursprung initial so zu wählen, dass aufschlussreiche Strukturen sichtbar werden, wird dem Benutzer selbst dann schwer fallen, wenn er bereits Vorkenntnisse vom betrachteten Strömungsfeld besitzt. Stattdessen wird der Anwender den Saatpunkt iterativ versetzen wollen, um einerseits eine Intuition für die zugrundeligenden Strömungsphänomene zu entwickeln und gleichzeitig eine informative Darstellung zu erzielen. Durch eine direkte Manipulation des Saatpunktes mit Hilfe der Maus kann dieses Vorgehen deutlich effizienter gestaltet werden, als es beispielsweise durch eine textuelle Eingabe der Raumkoordinaten möglich wäre.

6.4.3 Interaktionsmechanismen

Eine wissenschaftliche Simulationsumgebung versucht nicht zwangsläufig die Realität nachzubilden. Daher lässt auch die Umsetzung von Interaktionsmechanismen gewisse Freiheiten zu. Einige Modifikationsformen wie Rotation oder Translation lassen sich gut durch direktes Greifen des Objektes ermöglichen. In bestimmten Fällen ist es hingegen sinnvoll, die Interaktion durch zusätzliche Werkzeuge zu unterstützen. Zu diesem Zweck werden häufig die ursprünglich von Conner et al. [63] vorgestellten *Widgets* verwendet. Diese werden zusätzlich zu den eigentlichen Objekten dargestellt und sind häufig mit speziellen Griffen versehen, um eine Manipulation zu erleichtern. Änderungen, die der Anwender an den *Widgets* vornimmt, werden analog auch auf die verknüpften Objekte angewendet. Im Gegensatz zu der direkten Interaktion mit den Objekten selbst bieten *Widgets* einige Vorteile. So verfolgen sie beispielsweise ein einheitliches Bedienkonzept und können prinzipiell auch auf mehrere Objekte gleichzeitig wirken. Zusätzlich dazu bieten sie eine beständige Darstellung, die sich auch dann nicht ändert, wenn die Visualisierung in einem nachfolgenden Zeitschrittintervall aktualisiert wird. Die Interaktionsmechanismen können sich jedoch auch zu einem ernsthaften Problem entwickeln. Wenn mehrere Objekte dargestellt werden, kann es nämlich leicht geschehen, dass sich die Interaktion ungewollt auf ein falsches Objekt auswirkt [45]. Um dem vorzubeugen, werden die Interaktionsmechanismen selektiv nur für einzelne Objekte aktiviert. Die Aktivierung ist dabei abhängig von dem jeweiligen Kontext des in der Projekthierarchie ausgewählten Elements (vgl. Abschnitt 4.3.3). Die in VIRTUALFLUIDS INTERACTIVE zur Verfügung stehenden Interaktionsmechanismen sind exemplarisch in den Abbildung 6.11 bis 6.13 dargestellt.

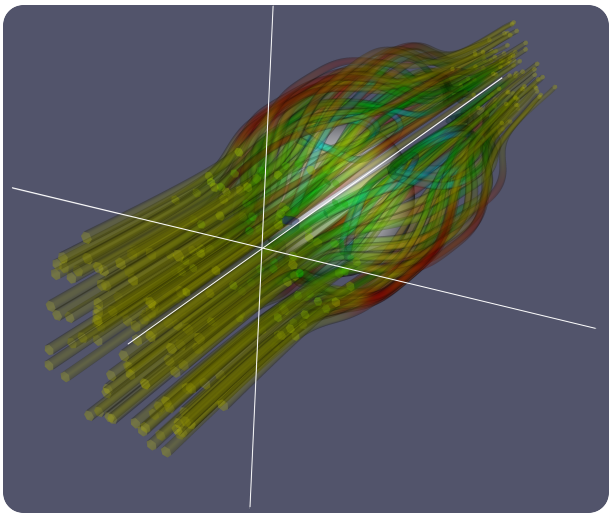


Abbildung 6.11: *Setzen des Saatpunktes von Strömungslinien mit Hilfe eines Fadenkreuzes.*

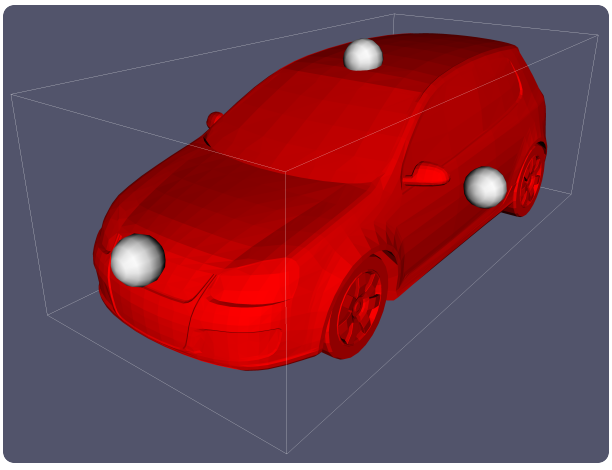


Abbildung 6.12: *Manipulation eines Objekts mit Hilfe eines Box-Widgets.*

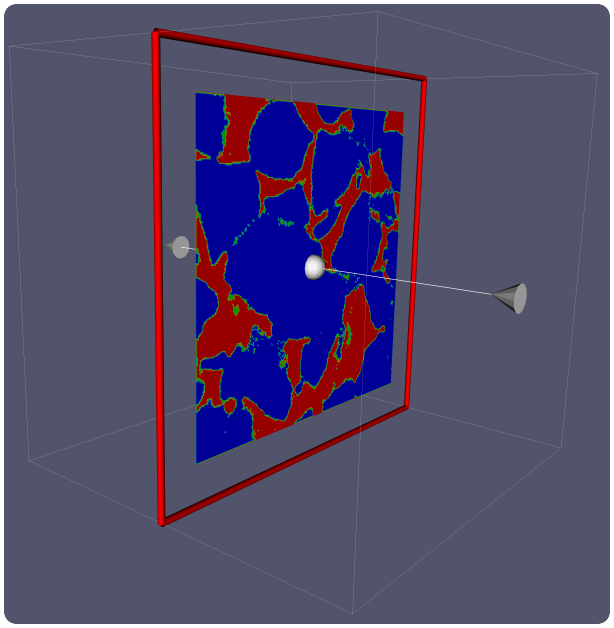


Abbildung 6.13: *Auswahl der Schnittebene mit Hilfe eines Plane-Widgets. Dargestellt ist der Schnitt durch ein poröses Medium.*

6.4.4 3D User-Interface-Komponenten

VIRTUALFLUIDS INTERACTIVE erlaubt zur Unterstützung einer intuitiven und effektiven Bedienung wann immer sinnvoll eine direkte grafische Manipulation. Allgemein üblich sind Interaktionsmechanismen zur Anpassung der Visualisierung von Ergebnisdaten. Mulder und van Wijk [196] stellen hingegen erstmalig einen Ansatz vor, der direkt manipulierbare grafische Elemente der Visualisierung zur Steuerung interaktiver Simulationen einsetzt. Sie führen parametrisierte Geometrieobjekte (*Parametrized Geometric Objects*) ein, deren Attribute an steuerbare Größen der Simulation gekoppelt werden können. Dieser Ansatz wird in VIRTUALFLUIDS INTERACTIVE aufgegriffen und erweitert. Im Gegensatz zu dem Verfahren von Mulder et al. wird dabei jedoch ein rein objektorientiertes Vorgehen verfolgt, das sich am Entwurf klassischer Benutzeroberflächen orientiert. Dafür werden zunächst wiederkehrende grafische Elemente identifiziert, die einerseits zur traditionellen Darstellung von Ergebnisdaten eingesetzt werden, andererseits aber auch unterschiedliche Typen der Modellhierarchie repräsentieren können. So kann ein Quader beispielsweise sowohl ein Element des geometrischen Modells darstellen, als auch einen Ausschnitt des Ergebnisraumes kennzeichnen, um die Menge der Ergebnisdaten zu reduzieren. Generell wird für jedes dieser Elemente eine komplexe Visualisierungspipeline aufgebaut, die im Kontext von VTK mindestens aus einem *Source*-Objekt, *Mapper*, *Actor* und einem *Widget* für die Interaktion besteht. Eine besondere Herausforderung besteht zusätzlich darin, die traditionell ausgabeorientierte Visualisierungspipeline zu invertieren und eine bidirektionale Bindung von Ein- und Ausgangsdaten zu erzielen. Um diese Komplexität vor dem Entwickler zu verbergen, wird eine zusätzliche Abstraktionsebene eingeführt. Analog zu klassischen komponentenbasierten Benutzerschnittstellen werden dafür generalisierte Komponenten definiert, die wiederkehrende dreidimensionale grafische Elemente sowohl visuell repräsentieren als auch eine Benutzerinteraktion ermöglichen. Die in VIRTUALFLUIDS INTERACTIVE als Grafikobjekte (*Graphic Objects*) bezeichneten 3D-Komponenten kapseln dabei eine vollständige Visualisierungspipeline. Diese Komponenten werden zusätzlich nach Quellen (*Sources*) und Filtern kategorisiert (vgl. Abbildung 6.14). Quellen repräsentieren Elemente der Modellhierarchie und Ergebnisdaten wie beispielsweise Kugeln, Dreiecksnetze oder Vektorfelder. An diese können Filter angefügt werden, um eine veränderte Sicht auf die Daten zu erhalten. So kann etwa ein Konturfilter eingesetzt werden, um die Isofläche eines bestimmten Wertes innerhalb eines Skalarfelds darzustellen. Ebenso wie Elemente klassischer Benutzeroberflächen erlauben Grafikobjekte eine individuelle Parametrisierung durch Vorgabe bestimmter Attribute oder Aktivierung bzw. Deaktivierung von Interaktionsmechanismen. Die Vertrautheit für den Entwickler wird zusätzlich durch einen Entwurf unterstützt, der sich an dem *Model View Presenter*-Muster, bekannt aus Abschnitt 4.3.2, orientiert. Auf diese Weise wird, wie in Abbildung 6.15 veranschaulicht, eine klare Trennung der Anliegen erreicht. Während die Grafikkomponenten im MVP-Muster dem *View* entsprechen, repräsentiert das *Model* traditionell Elemente des Domänenmodells. *Controller* korrespondieren mit *Presentern* und vermitteln zwischen Modell- und Grafikobjekten, indem sie eine bidirektionale Bindung herstellen, die Änderungen des Modells auf die Grafikobjekte überträgt und umgekehrt. Um dies zu erreichen, wird ein Benachrichtigungsmechnismus basierend auf dem Besuchermuster [97] eingesetzt. Zu beachten ist, dass keine direkte Abhängigkeit zwischen den Grafik- und Modellobjekten besteht, sondern jedwede Kommunikation ausschließlich über die *Controller* erfolgt. Mit deren Hilfe lassen sich Grafikobjekte sehr einfach an beliebige Instanzen des Domänenmodells binden. Das Resultat ist ein einheitliches und flexibles Entwurfskonzept für 3D-Grafikkomponenten, das für neue Elemente einfach zu erweitern und zu adaptieren ist. Erreicht wird

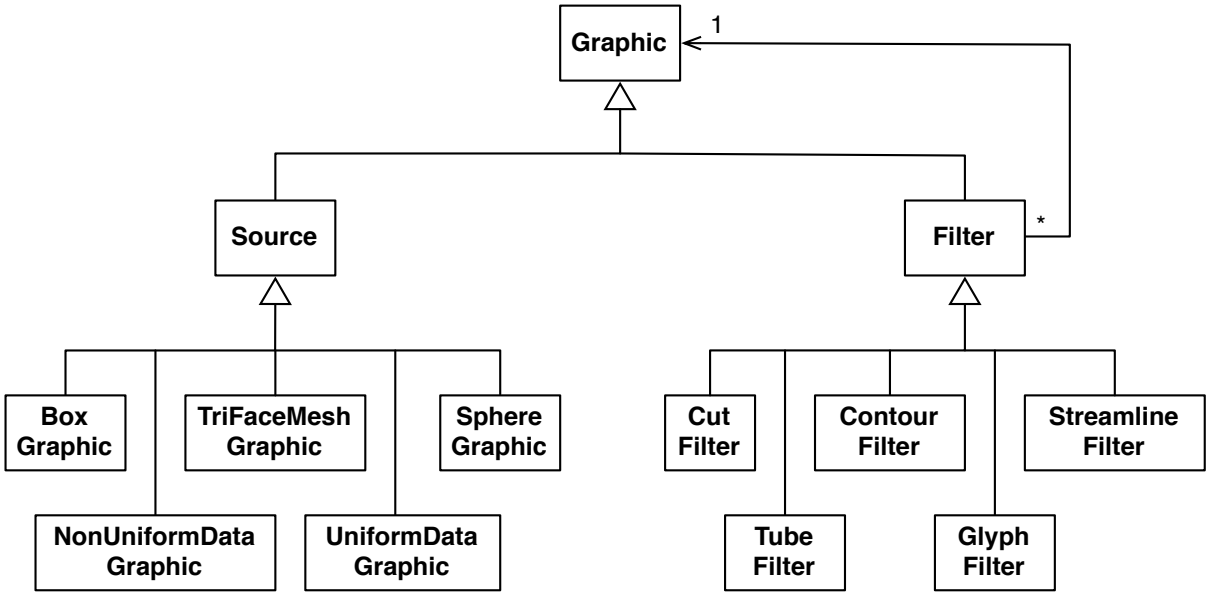


Abbildung 6.14: *Darstellung der Hierarchie grafischer Elemente.*

dies durch eine lose Kopplung von Grafik- und Modellobjekten mit Hilfe intermediärer *Controller*, die eine klare Trennung von Verantwortlichkeiten vorsieht. Der entstehende Entwurf fördert zugleich die Wiederverwendung und das Testen einzelner Module. Zusätzlich ermöglicht er, flexibel auf Änderungen zu reagieren. Im Extremfall lässt sich beispielsweise die Ebene der 3D-Visualisierung durch Neuimplementierung der Grafikobjekte auf eine andere Grafikbibliothek umstellen, ohne Änderungen an den übrigen Ebenen zu erfordern.

6.4.5 Responsivität

Eine interaktive Simulationsumgebung wird auf eine Benutzerinteraktion stets mit einer gewissen Latenz reagieren. Diese wird in der Regel von der Laufzeit der beteiligten Prozesse bestimmt. In VIRTUALFLUIDS INTERACTIVE sind an dieser Stelle vor allem die Laufzeiten der Visualisierung und der Simulation hervorzuheben. Eine zu lange Antwortzeit des Systems wird dabei in bestimmten Fällen von dem Benutzer als hinderlich empfunden. So wird ein Anwender beispielsweise große Schwierigkeiten haben, den Saatpunkt einer Stromlinie festzulegen, wenn sich das dazu verwendete *Widget* mit einer zu starken Verzögerung bewegt. Vielmehr sollte die Latenz in einem solchen Szenario nicht länger als 0,1 Sekunden betragen [45, 152, 251]. Eine demgegenüber verzögerte Aktualisierung der Visualisierung wird sich hingegen nicht grundsätzlich störend auf den Interaktionsprozess auswirken. Aus diesem Grund werden die Aktualisierung der Visualisierung und die Benutzerinteraktion in VIRTUALFLUIDS INTERACTIVE separiert verarbeitet, um hinderlichen Latenzen während der Interaktion vorzubeugen.

Neben der Laufzeit komplexer Visualisierungsalgorithmen kann auch die Interaktion mit der Simulation zu relevanten Verzögerungen führen. Dabei besteht ein grundsätzliches Problem darin, dass Anwender bei einer Antwortzeit von mehreren Sekunden dazu neigen anzunehmen, das System habe ihre Eingabe nicht registriert oder sei gar abgestürzt [204]. Dieses Verhalten führt letztlich dazu,

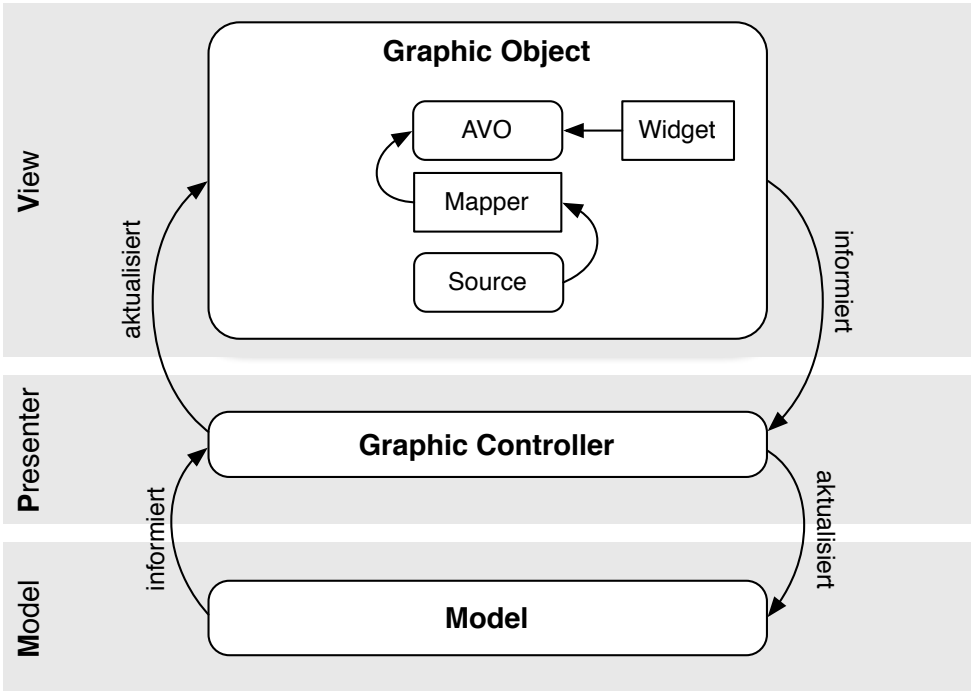


Abbildung 6.15: *Diagram kollaborierender Komponenten im Kontext der Visualisierung.*

dass die Eingabe wiederholt wird, wodurch die Last auf das System und damit die Verzögerung noch gesteigert wird. Wright nimmt sich dieses Problems an und erläutert, dass Anwender basierend auf ihrer Eingabe, der zu erwartenden Reaktion des Systems und der Latenz ein mentales Modell ausbilden [312]. Entsprechend dieser Vorstellung bereiten dem Anwender konsistente Latenzen wenig Schwierigkeiten. Ein Problem entsteht jedoch bei variablen Antwortzeiten.

In VIRTUALFLUIDS INTERACTIVE werden Steuerungsbefehle des Anwenders grundsätzlich unmittelbar ausgeführt. Insofern sie keinen zusätzlichen Rechenaufwand erfordern, beeinflussen die Befehle dabei nicht die Laufzeit des aktuellen Simulationsintervalls. Der Anwender wird die Auswirkung seiner Eingaben somit mit der folgenden Aktualisierung der Ergebnisdaten in konstantem Zeitintervall beobachten können. Die Antwortzeit T_R auf eine Benutzerinteraktion ist dabei wie folgt limitiert:

$$T_R \leq T_{\text{sim}} + T_{\text{sim}\rightarrow\text{vis}} + T_{\text{vis}} + T_{\text{ren}} \tag{6.3}$$

Eine andere Situation entsteht, wenn in Folge der Benutzerinteraktion laufzeitintensive Prozesse angestoßen werden, die zusätzliche Latenzen verursachen. Dazu gehören Änderungen am geometrischen Modell, die ein erneutes Generieren des Berechnungsgitters erfordern. Diese zusätzlichen Verzögerungen resultieren letztlich in einer Änderung des Aktualisierungsintervalls und können damit zu einer Desorientierung des Anwenders führen. Wright empfiehlt in dieser Situation, den Anwender bewusst auf den andauernden Vorgang aufmerksam zu machen und ihn auf diese Weise davon abzuhalten, den Befehl zu wiederholen [312]. In VIRTUALFLUIDS INTERACTIVE wird dieses Vorgehen dadurch erreicht, dass die entsprechenden Schaltflächen und Menüeinträge zum Ausführen des Befehls bis zum Ende des aktiven Vorganges deaktiviert werden und damit ein Andauern des Prozesses signalisieren.

6.5 Visualisierung großer Datenmengen

Seit Beginn des Computerzeitalters erlebt die Welt ein explosionsartiges Wachstum an Informationen, das maßgeblich auf den exponentiellen Leistungsanstieg bei Prozessoren zurückzuführen ist (vgl. Mooresches Gesetz, Abschnitt 5.2) [68, 115, 135, 173, 197]. Mehr als 90% der seit 2003 neu geschaffenen Daten liegen dabei mittlerweile in digitaler Form vor [171]. Eine der großen Herausforderungen des 21. Jahrhunderts wird darin bestehen, diese fulminante Datenflut effektiv zu verarbeiten und die enthaltenen Informationen zugänglich zu machen [197]. Dabei werden Techniken der Visualisierung eine entscheidende Rolle einnehmen.

Die kontinuierlich steigende Leistungsfähigkeit von Hochleistungsrechnern ermöglicht auch im Bereich numerischer Strömungssimulationen immer größere Systeme und bedingt damit gleichzeitig ein stetiges Wachstum der Ergebnisdatensätze. Schon heute umfasst die Größe dieser Daten leicht mehrere Terabytes und es ist abzusehen, dass schon bald die Petabytegrenze überschritten werden wird. Die Verarbeitung und Analyse dieser kontinuierlich wachsenden Datenmengen stellt sowohl für traditionelle, im *Post-Processing*-Verfahren arbeitende Simulationssysteme als auch für interaktive Ansätze eine zunehmend schwierige Aufgabe dar. Von erheblicher Bedeutung ist dabei die erhöhte Last auf der Visualisierungspipeline. Ein typischer Ansatz zur Balancierung dieser Last besteht darin, die Analyse des Ergebnisraums auf einen Ausschnitt zu begrenzen (*Cropping*) und/oder die Ergebnisse zeitlich und räumlich auszudünnen (*Subsampling*). Dieses Vorgehen findet auch in VIRTUALFLUIDS INTERACTIVE Anwendung und wurde in Teilen bereits in Abschnitt 6.3.3 diskutiert. In Abschnitt 5.2 wurde überdies erläutert, dass insbesondere Grafik- und Visualisierungssysteme überdurchschnittlich von den technologischen Entwicklungen profitieren und somit in der Lage sind, zunehmend komplexere Darstellungen von Massendaten zu erzeugen. Die Komplexität der grafischen Repräsentation wird letztlich jedoch von den menschlichen Fähigkeiten der visuellen Wahrnehmung und Kognition limitiert, deren Kapazität von Natur aus konstant ist und somit dem Gesetz von Moore nicht genügen kann.

6.5.1 Stereoskopische Darstellung

Um dem menschlichen Faktor gerecht zu werden, ist es nicht ausreichend, die Leistungsfähigkeit der Systeme stetig zu verbessern. Vielmehr sollten sich diese vermehrt an menschlichen Charakteristiken, Stärken und Limitierungen orientieren [197]. Ein Lösungsansatz kann beispielsweise darin bestehen, die metaphorisch ausgedrückte Bandbreite der Wahrnehmungsfähigkeit zu erhöhen, indem neben dem visuellen Sinn auch weitere Fähigkeiten wie beispielsweise die taktile und kinästhetische Wahrnehmung eingebunden werden. So besteht ein häufiges Problem der visuellen Darstellung CFD-typischer Volumendaten in der gegenseitigen Überdeckung einzelner Elemente. Geräte zur haptischen Darstellung unterliegen diesen Einschränkungen demgegenüber nicht, so dass der Analyseprozess effizienter gestaltet werden kann. Eine weitere Möglichkeit zur Verbesserung der Wahrnehmung ist der Einsatz einer stereoskopischen Darstellung, die dem Anwender einen realistischen Tiefeneindruck suggeriert. So sind in die Entwicklung von VIRTUALFLUIDS INTERACTIVE die Erfahrungen früherer Arbeiten von Fahrig [84], Linxweiler [166] und Mittelstaedt [193] eingeflossen, die sich unter anderem mit der immersiven 3D-Darstellung und interaktiven Exploration von Ergebnisdaten aus Strömungssimulationen innerhalb einer *Virtual-Reality*-Umgebung befassen. In [166]

wird der Benutzer dabei in die Lage versetzt, den Ergebnisraum aktiv zu erkunden und sich in der virtuellen Umgebung zu bewegen. Dazu wird die Position und Blickrichtung des Benutzers und eines Eingabemediums ermittelt. Der Grundgedanke dabei ist, den Anwender weitestgehend in den Explorationsprozess zu involvieren, um dadurch die Wahrnehmung und letztlich das Verständnis zu fördern [44, 68].

Am Institut für rechnergestützte Modellierung im Bauingenieurwesen (iRMB) steht dafür ein VR-Labor zur Verfügung, dass unter anderem mit einem System zur passiven Stereoprojektion sowie einem *Tracking*-System zur Positionsbestimmung ausgestattet ist (vgl. Abbildung 6.16). Obwohl es sich

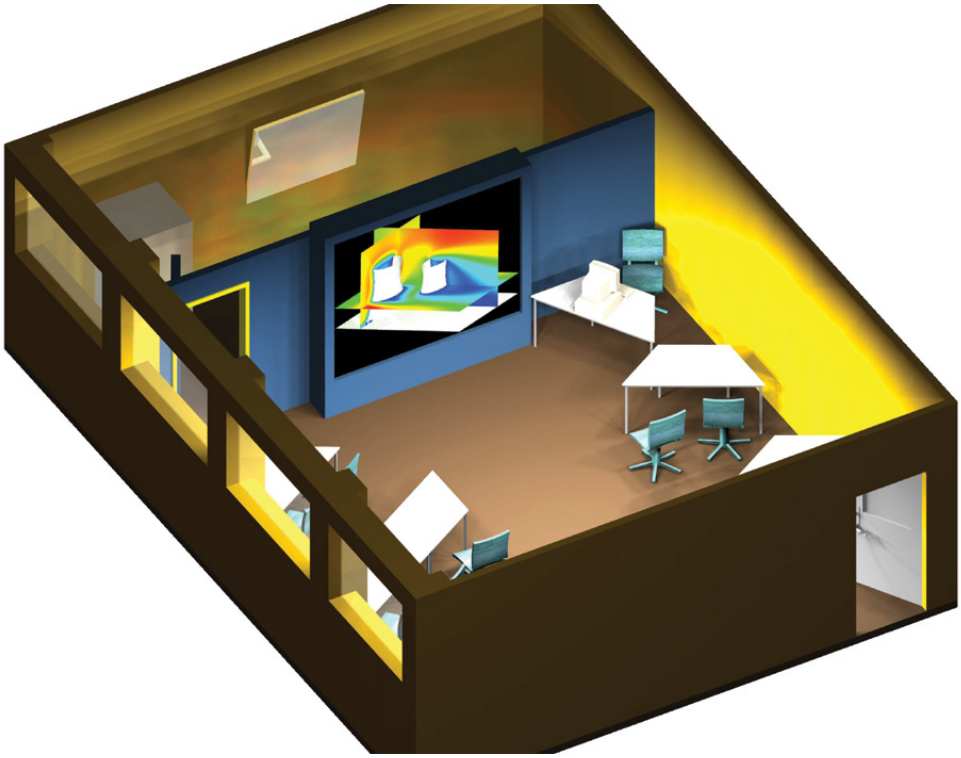


Abbildung 6.16: *VR-Labor des iRMB mit Leinwand für passive Stereoprojektion und Tracking-System [84].*

um eine vielversprechende Technologie handelt, sind VR-Systeme bislang nur begrenzt verbreitet. Es ist anzunehmen, dass ein Hauptgrund dafür in den hohen Anschaffungskosten besteht [17]. Nachteilig sind gleichzeitig der erhebliche Raumbedarf und der vergleichsweise aufwändige Betrieb. Vor diesem Hintergrund wurde der Ansatz im Rahmen dieser Arbeit zu Gunsten einer preiswerten und komfortablen Desktoplösung zunächst nicht weiter verfolgt. Die aktuellen Entwicklungen auf dem Gebiet der stereoskopischen Darstellung im Bereich der Filmindustrie und Fernsehgeräte lassen jedoch hoffen, dass die Technologie in Zukunft weitere Verbreitung finden wird.

6.5.2 Langfristige Perspektive

In Abschnitt 2.4 wurden bereits Gründe dafür erläutert, weshalb die meisten Simulationen nach wie vor nicht interaktiv sondern traditionell im *Post-Processing*-Verfahren ausgewertet werden. Dieses Vorgehen erfordert unter anderem ein dauerhaftes Speichern von Ergebnisdaten aus der lau-

fenden Simulation heraus. Das Persistieren dieser stetig wachsenden Datenmenge stellt jedoch eine zunehmende Herausforderung dar. Bereits heute werden bis zu 90% der berechneten Daten ohne vorherige Analyse schlicht verworfen [135], da die Wachstumsrate nichtflüchtiger Speichergenerationen, allen voran der von Festplatten, dem exponentiell zunehmenden Datenwachstum nicht standhält [135, 318]. Ein zusätzliches, ebenso gravierendes Problem, besteht speziell im CFD-Umfeld darin, dass großskalige Simulationen zwar auf leistungsstarken Supercomputern oder Clustersystemen durchgeführt werden, die Verarbeitung der Ergebnisdaten hingegen häufig auf dem lokalen System des Anwenders erfolgt [189]. Wie bereits in Abschnitt 6.3.3 diskutiert, werden dabei die zur Analyse eingesetzten Visualisierungssysteme auf Grund der immensen Datenmengen stetig an ihre Leistungsgrenzen getrieben, weshalb eine langfristige Lösung für dieses Problem aus Sicht von Visualisierungsexperten nur in einer Annäherung von Simulation und Analyse bestehen kann [68, 173, 211, 318] (vgl. In-Situ-Visualisierung, Abschnitt 6.2.2). Werden Analyse und Visualisierung parallel mit der Simulation auf den Hochleistungssystemen ausgeführt, kann somit der aufwändige Transfer von rohen Massendaten entfallen, während gleichzeitig die Visualisierung erheblich von der leistungsfähigen HPC-Umgebung profitiert. Die resultierende Menge an aufbereiteten Daten ist dabei, verglichen mit den Rohdaten, deutlich geringer, so dass sowohl der Bedarf an Speicherplatz als auch an Bandbreite für ein mögliches *Monitoring* gesenkt wird. Obgleich vorteilhaft gegenüber traditionellen Vorgehensweisen, wird dieser Ansatz dennoch aus zwei maßgeblichen Gründen nur selten verfolgt. Erstens sind die Anwender von Hochleistungsrechnern unwillig, teure Rechenzeit für Visualisierungen aufzuwenden. Dies gilt vor allem für aufwändige Visualisierungsalgorithmen, die ungünstigstenfalls die Gesamtlaufzeit der Simulation übersteigen. Die notwendige Kompetenz vorausgesetzt, ist die Kopplung mit der Visualisierung zweitens oft mit einem erheblichen Mehraufwand verbunden [173].

6.5.3 Mittelfristiger Lösungsansatz auf Basis von HDF5

Im Rahmen dieser Arbeit wird daher zunächst ein mittelfristiger Lösungsansatz verfolgt, der in erster Linie eine gängige Limitierung sowohl frei verfügbarer als auch kommerzieller Visualisierungsumgebungen adressiert. Viele der durchaus populären *Post-Processing*-Systeme wie beispielsweise Paraview [144] und AVS [3] stoßen bei der Verarbeitung von Massendaten schnell an Ressourcengrenzen, da sie stets und ausschließlich vollständige Datensätze einlesen und im Speicher vorhalten. VIRTUALFLUIDS INTERACTIVE umfasst daher eine Erweiterung um zusätzliche *Post-Processing*-Funktionalität und spezifiziert ein Format für den Datenaustausch zwischen Simulation und Visualisierungsumgebung auf Basis von HDF5 [275], mit dessen Hilfe die Anforderungen umgesetzt werden können. HDF5 ist gleichzeitig sowohl ein Datenformat als auch eine Bibliothek zur flexiblen und effizienten Verwaltung komplexer Massendaten mit Schnittstellen zu verschiedenen Programmiersprachen wie C++, Fortran und Java. Entwickelt und bereitgestellt wird HDF5 von dem US-amerikanischen National Center for Supercomputing Applications (NCSA). Um Entwicklern eine einheitliche und komfortable Schnittstelle zu bieten, wird die Komplexität und Funktionsvielfalt der Bibliothek in VIRTUALFLUIDS INTERACTIVE in einer Komponente mit anwendungsspezifischer Funktionalität verborgen (vgl. Fassadenmuster [97]), die es ermöglicht, sowohl uniforme als auch blockstrukturierte Ergebnisdaten aus einer laufenden Simulation heraus zu persistieren. Die Unterstützung von verfeinerten, blockstrukturierten Gittern erlaubt dabei die Anbindung des am iRMB entwickelten CPU-basierten nicht-uniformen LB-Lösers, der von Freudiger [94] und Geller [104] beschrieben

wird. Jedes Gitter kann dabei mehrere Gitterstufen unterschiedlicher Auflösungen enthalten, um in bestimmten Bereichen eine Verfeinerung zu erreichen. Dabei werden die Knoten eines Gitterlevels jeweils in sogenannte Blöcke von einheitlicher Dimension eingeteilt (Hybridgitter). Diese bilden gleichsam die Basis der Schnittstelle zur Persistierung des Gitters. Ein Block ist per Definition immer voll besetzt. Das Blockgitter hingegen kann Bereiche enthalten, in denen keine Blöcke oder Blöcke eines anderen Gitterlevels vorliegen. Diese Bereiche können im HDF5-basierten Format durch sogenannte *Chunks* abgebildet werden. Für einen *Chunk* wird nur dann Speicher reserviert, wenn er auch vorhanden ist. Pro Gitterlevel und makroskopischer Größe wird folglich eine uniforme Matrix bestehend aus einzelnen *Chunks* vorgesehen, für die erstmals Speicher reserviert wird, sobald der entsprechende Block geschrieben wird. Dabei können die Daten gleichzeitig komprimiert werden, um den Speicherbedarf zu reduzieren.

Die Ergebnisdaten, basierend auf dem beschriebenen Format, können anschließend im *Post-Processing*-Verfahren von VIRTUALFLUIDS INTERACTIVE verarbeitet werden. Dabei unterstützt die Anwendung das in Abschnitt 6.3.3 erörterte Mantra der Informationssuche, so dass Ressourcen und Detaillierungsgrad feingranular balanciert werden können. Das Einlesen der Ergebnisdaten kann indes sowohl auf einen bestimmten Ausschnitt begrenzt werden (*Cropping*), als auch hinsichtlich der räumlichen Auflösung reduziert werden (*Subsampling*). Gleichzeitig ist es möglich, das *Post-Processing* auf Bereiche bestimmter Gitterlevel einzugrenzen, wobei Werte zwischen feinen und groben Stufen bei Bedarf interpoliert werden, um eine kontinuierliche Darstellung zu garantieren. Abbildung 6.17 zeigt beispielhaft die Umströmung einer Kugel berechnet auf einem blockstrukturierten Gitter mit drei Verfeinerungsstufen und einer Gesamtknotenanzahl von 20 Millionen.

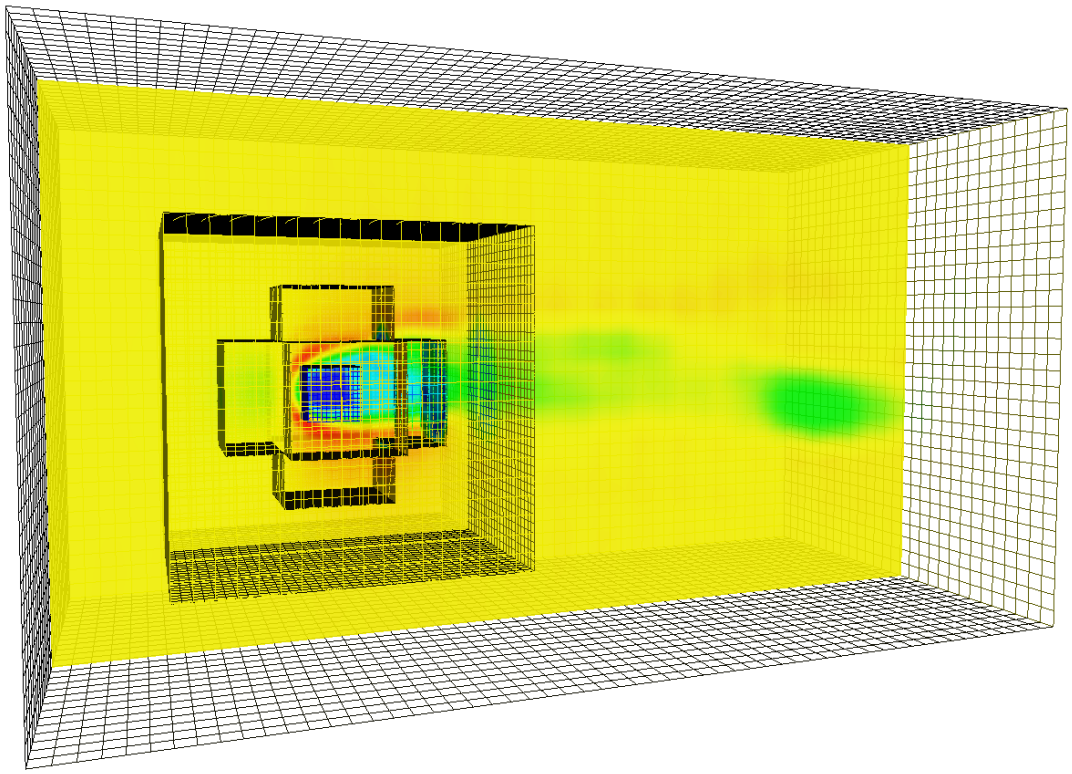


Abbildung 6.17: *Post-Processing blockstrukturierter Ergebnisdaten in* VIRTUALFLUIDS INTERACTIVE.

7 Anwendungsbeispiele und Validierung

> »But what is it good for?«[1]
> Kommentar eines Ingenieurs der Advanced Computing Systems Division von IBM zum Mikrochip, 1968

In diesem Kapitel werden einige ausgewählte Anwendungsbeispiele aus dem Gebiet des Bauingenieurwesens vorgestellt. Sie sollen aufzeigen, dass die computergestützte Simulation im Bereich der Strömungsmechanik als wertvolles Hilfsmittel für Ingenieure und Wissenschaftler dienen kann, das den Planungsprozess sinnvoll unterstützt. Insbesondere soll jedoch darauf aufmerksam gemacht werden, dass sich die notwendigen Werkzeuge dank der Leistung moderner Grafikprozessoren in die gewohnte Arbeitsumgebung integrieren lassen, ohne bedeutende finanzielle Investitionen oder interdisziplinäre Kompetenzen zu erfordern.

7.1 Windinduzierte Interferenz bei Gebäuden

Windlasten stellen eine wichtige Bemessungsgröße für den Entwurf von Bauwerken dar. Bei der zunehmend dichteren Bebauung heutiger Städte ist dabei der wechselseitige Einfluss von Gebäuden von wachsender Bedeutung. Dies gilt besonders für Bauwerke mit großen Gebäudehöhen wie beispielsweise Hochhäusern, die in der heutigen Zeit stetig höher geplant und gebaut werden. Die dabei auftretende wechselseitige Beeinflussung wird in der Literatur als windinduzierte Interferenz oder auch Windinterferenz bezeichnet [288].

Bailey und Vincent [14] betrieben bereits während des 2. Weltkrieges im Jahre 1943 Grundlagenforschung auf diesem Gebiet, während dem Phänomen in der Folgezeit wenig Aufmerksamkeit gewidmet wurde. Im Fokus der Forschungsarbeit hinsichtlich Windeinflüssen stehen vielmehr die Auswirkungen auf isolierte Gebäude, da allgemein von der Annahme ausgegangen wird, dass sich Windlasten auf Grund von Abschirmungseffekten in Gruppenanordnungen tendenziell reduzieren [287]. Mit dem Unglück im englischen Ort Ferrybridge zu Beginn der 70er Jahre rückte die Thematik erneut ins Interesse und erfährt seitdem eine Wiederaufnahme. Dort kam es 1965 in Folge starker Winde zum Einsturz von drei in Reihe stehenden Kühltürmen. Die Ursache dafür wurde in windinduzierten Interferenzen gefunden [10]. Wie sich herausstellte, war die Belastung auf die eingestürzten Kühltürme drei mal höher als bei der Bemessung angenommen, obwohl sie auf der windabgewandten Seite hinterliegend in zweiter Reihe errichtet wurden. Das Unglück bleibt kein Einzelfall. Vielmehr kommt es immer wieder zu windinduzierten Schäden an Bauwerken [270], die zum Teil auf Windinterferenzen zurückzuführen sind.

Die Bemessung der Windlasten erfolgt in Deutschland in der Regel nach den allgemein anerkannten Regeln der Technik, die beispielsweise im Eurocode EN 1991-1-4 [71] erläutert werden. Dabei ist

[1] »Aber wofür ist das gut?«

Abbildung 7.1: *Einsturz mehrere Kühltürme im englischen Ferrybridge in Folge von Windinterferenz [203].*

grundsätzlich auch der Einfluss vorhandener Bebauung mit zu berücksichtigen. Khanduri [142] gibt jedoch zu bedenken, dass dem Phänomen windinduzierter Interferenzen nach wie vor auch in der aktuellen Normung wenig Aufmerksamkeit gewidmet wird. So bleibt der Einfluss neuer Bebauung auf bereits bestehende Gebäude von der Norm gänzlich unbeachtet. Uffelen [288] begründet diesen Umstand mit der Komplexität des Phänomens, das mit vereinfachten Regeln nicht hinreichend genau beschrieben werden kann. Des Weiteren finden sich in der wissenschaftlichen Literatur nur vereinzelt experimentell ermittelte Datenbestände [142, 159, 238, 273], so dass eine Ableitung belastbarer Entwurfsgrößen diffizil ist. Insbesondere bei außergewöhnlichen Gebäudeformen bleibt als einzige Alternative häufig nur eine kostenintensive experimentelle Untersuchung in einem Windkanal. Als Alternative bzw. Unterstützung des teuren Experiments wird an dieser Stelle daher eine Erweiterung von VIRTUALFLUIDS INTERACTIVE vorgestellt, die als Bemessungshilfe für Windinterferenzen bei Gebäuden in Gruppenanordnung auf Simulationsbasis dient.

7.1.1 Ermittlung des Interferenzfaktors

Windinterferenzen werden im Allgemeinen durch einen sogenannten Interferenzfaktor (IF) quantifiziert, der wie folgt definiert ist [142]:

$$IF = \frac{\text{Windlast auf ein Gebäude in Gruppenanordnung}}{\text{Windlast auf ein isoliertes Gebäude.}} \tag{7.1}$$

Generell können die Windlasten dabei sowohl mittleren Kräften oder Momenten auf die Haupttragstruktur entsprechen oder auch lokalen Spitzenlasten, die beispielsweise auf die Fassade wirken. Die Erweiterung von VIRTUALFLUIDS INTERACTIVE beschränkt sich bei der Ermittlung das Interferenzfaktors jedoch zunächst auf die mittlere Belastung in Folge des Windwiderstandes auf die Gesamtstruktur des Gebäudes.

Die Berechnung der mittleren einwirkenden Kraft auf ein festes Hindernis im Lattice-Boltzmann-Kontext erfolgt dabei entsprechend der Beschreibung von Nguyen und Ladd [201]. Danach berechnet sich die Kraft $\vec{F}_k$ aus dem Impulsaustausch der Massenanteile $f_{\hat{i}}(t,\vec{x}_f)$ und $f_i(t+\Delta t,\vec{x}_f)$ an dem Schnittpunkt der Hindernisstruktur mit der Verbindungslinie zwischen den Knoten $\vec{x}_f$ und $\vec{x}_b$ (vgl. Abbildung 7.2). Die Berechnung des Momentenaustauschs berücksichtigt dabei die Massenteile vor

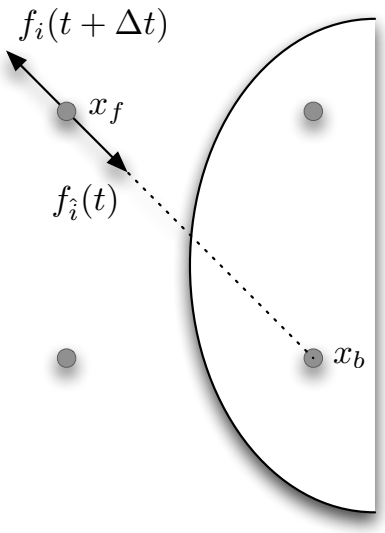


Abbildung 7.2: *Impulsaustausch an festen Hindernissen.*

und nach dem Auftreffen auf das Hindernis:

$$\vec{F}_k(t+\Delta t/2) = -\frac{V}{\Delta t}\vec{e}_i\left(f_i(t+\delta t,\vec{x}_f) + f_{\hat{i}}(t,\vec{x}_f)\right), \tag{7.2}$$

wobei V dem Volumen einer Einheitszelle entspricht und $\hat{i}$ die Gegenrichtung zu i beschreibt ($\vec{e}_i = -\vec{e}_{\hat{i}}$). Die gesamte Krafteinwirkung auf ein Hindernis ergibt sich gleichsam aus der Summe aller beitragenden $\vec{F}_k$.

$$\vec{F} = \sum_{k\in\mathfrak{C}} \vec{F}_k \tag{7.3}$$

$\mathfrak{C}$ entspricht dem Satz aller von dem Hindernis geschnittenen Knotenverbindungen, wobei ausschließlich Randknoten $\vec{x}_f$ betrachtet werden.

Analog zu der Berechnung der übrigen makroskopischen Größen erfolgt die Ermittlung der Kräfte auf die festen Hindernisstrukturen in der Implementierung innerhalb einer separaten *Kernel*-Funktion. Von dieser Routine wird die Einwirkung zunächst auf Basis einzelner Knoten bestimmt, die anschließend zu einer Gesamteinwirkung in Bezug auf einzelne Hindernisstrukturen aufsummiert werden. Einzelne Knotenwerte können den jeweiligen Hindernissen dabei mit Hilfe der Voxelbeschreibung zugeordnet werden. Zu diesem Zweck wird die Geometriebeschreibung der Knoten um eine eindeutige Bezeichnung der abgebildeten Hindernisstruktur ergänzt, ohne jedoch zusätzlichen Speicherplatz zu beanspruchen. Stattdessen wird die Information unter Verwendung von *Bit-Shift*-Operationen mit in die bereits reservierten 8 Bit pro Knoten kodiert.

Die *Kernel*-Funktion für die Berechnung der Krafteinwirkung sieht korrespondierend zu dem in Abschnitt 5.3 beschriebenen Verfahren vor, jeden Knoten des Berechnungsgitters in einem separaten CUDA-*Thread* zu verarbeiten. Entsprechend wird an dieser Stelle das in Abschnitt 5.3.1 beschriebene Gridlayout verwendet. Algorithmus 7.1 beschreibt das grundlegende Vorgehen des lokalen Verfahrens. Zunächst wird der jeweilige Knotentyp ermittelt. Handelt es sich dabei um einen Hindernis-

Algorithmus 7.1 Lokale Kraftberechnung am Knoten(x,y,z)

Require: x, y, z
1: $\text{force}_x = \text{force}_y = \text{force}_z = 0.0$ ▷ Lokale Kräfte in X-,Y-,Z-Richtung
2: **if** geo(x, y, z) == SOLID **then** ▷ Wenn Hindernisknoten dann...
3: **for all** d in 19 directions **do**
4: compute x_n, y_n, z_n for neighbor in direction d
5: **if** geo(x_n, y_n, z_n) == FLUID **then** ▷ Wenn Fluid-Nachbar...
6: compute partial forces f_x, f_y, f_z in direction d
7: add partial forces f_k to local force_k with $k \in \{x, y, z\}$
8: **end if**
9: **end for**
10: **end if**

knoten, der gleichsam einen Randknoten darstellt, wird die lokale Krafteinwirkung auf den Knoten in Richtung der benachbarten Fluid-Knoten bestimmt und gespeichert. In einem Folgeschritt werden die lokalen Kräfte zu einer Gesamteinwirkung aufsummiert. Für diese Aufgabe wird ein paralleler Algorithmus zur Berechnung von Präfixsummen [25], auch als Scan-Algorithmus bezeichnet, verwendet. Die GPU-Implementierung basiert dabei auf der Beschreibung von Harris [117]. Algorithmus 7.2 beschreibt das grundlegende Verfahren, das zur Veranschaulichung zusätzlich in Abbildung 7.3 grafisch dargestellt ist. Generell erfolgt die Summation separat für jede der drei Raumrichtungen. Dabei werden zunächst die Werte in X-Richtung aufsummiert und die Datenmenge auf zwei Dimensionen reduziert. Anschließend wird der Algorithmus in gleicher Weise in den beiden anderen Richtungen angewendet. Die Reduktion erfolgt schrittweise durch Bildung der Teilsummen aus jeweils zwei Elementen. Dadurch erfolgt in jedem Schritt eine Halbierung der entsprechenden Dimension. Die Variable *stride* entspricht indes jeweils der Hälfte der aktuellen Ausdehnung, so dass bei der Summation der i-te *Thread* jeweils das i-te und (*stride*+i)-te Element bearbeitet. Die Teilergebnisse werden anschließend an i-ter Position zwischengespeichert. (Der vorherige Eintrag an i-ter Stelle geht dabei verloren.) Da das Verfahren ein wiederholtes Schreiben und Lesen einzelner Elemente erfordert, werden die Summen im *Shared Memory* gebildet. Dieses Vorgehen wird so lange wiederholt, bis *stride*

Algorithmus 7.2 Paralleler Scan-Algorithmus

Require: x, y, z
Require: $\text{force}_x = \text{force}_y = \text{force}_z$
1: $\text{buffer}_x[x] = \text{force}_x$
2: $\text{buffer}_y[x] = \text{force}_y$
3: $\text{buffer}_z[x] = \text{force}_z$
4: stride = $x_{\text{max}} >> 1$
5: **while** stride > 0 **do**
6: __syncthreads()
7: **if** $x <$ stride **then**
8: $\text{buffer}_x[x] += \text{buffer}_x[x + \text{stride}]$
9: $\text{buffer}_y[x] += \text{buffer}_y[x + \text{stride}]$
10: $\text{buffer}_z[x] += \text{buffer}_z[x + \text{stride}]$
11: **end if**
12: stride >>= 1
13: **end while**

die Größe null erreicht hat. Anschließend liegt die Summe aller Werte in der jeweiligen Raumrichtung in dem ersten Eintrag vor. Der Algorithmus weist damit eine Laufzeitkomplexität von $\log(n)$ auf, erfordert dafür jedoch zwingend eine Ausdehnung von Größe einer Potenz zur Basis zwei. Dieser Anforderung kann beispielsweise durch Auffüllen mit Nullen nachgekommen werden (*Padding*).

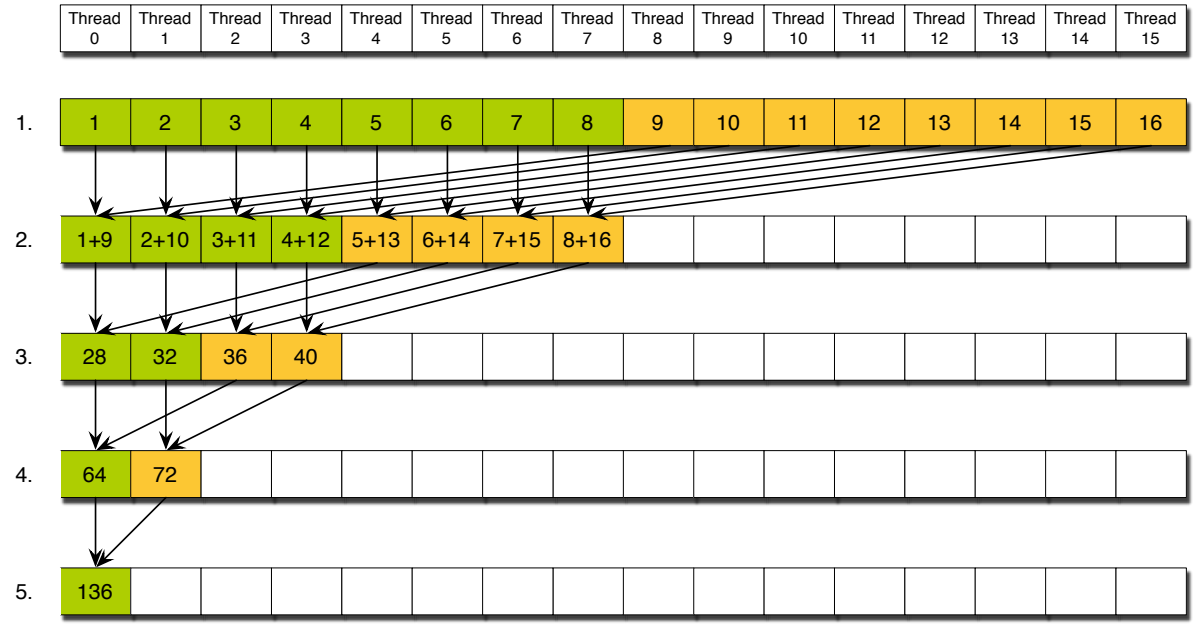


Abbildung 7.3: *Veranschaulichung des parallelen Scan-Algorithmus.*

7.1.2 Validierung

Die vorausgehend erläuterten Algorithmen ermöglichen es, die jeweilige Krafteinwirkung des Fluids auf beliebige Hindernisobjekte zu ermitteln und daraus den Interferenzfaktor für bestimmte Gebäude abzuleiten. Im Folgenden wird das beschriebene Verfahren mit Ergebnissen aus der Literatur verglichen, die verschiedenen experimentellen Untersuchungen entstammen. Khanduri [142] beschreibt detailliert einen entsprechenden Modellversuch im Windkanal, der analog in der Simulationsumgebung durchgeführt wird. Der grundlegende Versuchsaufbau ist schematisch in Abbildung 7.4 dargestellt. Der Aufbau umfasst zwei identische Gebäude im Maßstab 1 : 400, die in Reihe

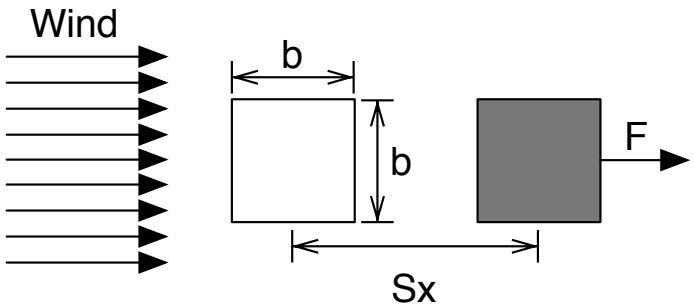


Abbildung 7.4: *Schematische Darstellung des Versuchsaufbaus zur Validierung.*

angeordnet sind und frontal angeströmt werden. Dabei wird der wechselseitige Einfluss der Gebäude auf die jeweilige mittlere Windbelastung (F) in Form des Interferenzfaktors (IF) ermittelt. Die Gebäudemodelle des Windkanals haben eine Seitenlänge $b = 5$ cm und eine Gebäudehöhe $h = 20$ cm. Sie werden mit einer Geschwindigkeit von 11 $\frac{m}{s}$ in Gebäudehöhe angeströmt, wobei das Einstromprofil nach [262] so gewählt wird, dass die umgebende Bepflanzung und Bebauung berücksichtigt wird

(vgl. Abbildung 7.5). Der vorliegende Anwendungsfall geht von einer urbanen Bebauung aus. Dar-

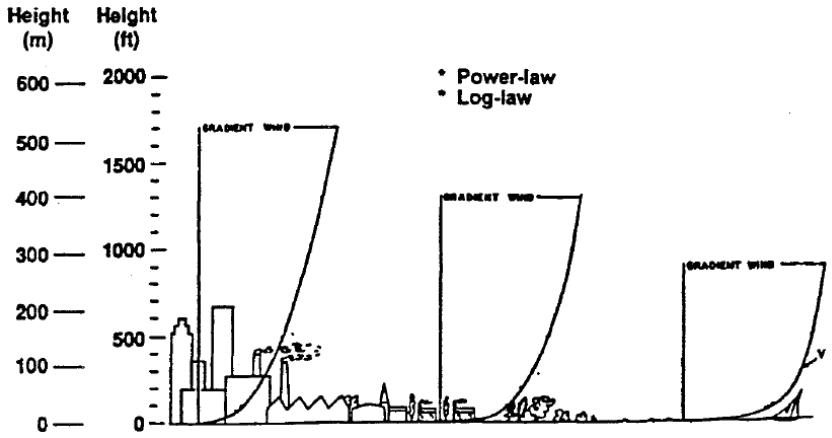


Abbildung 7.5: *Windgeschwindigkeitsprofile bei unterschiedlichen Geländeverhältnissen [262].*

aus ergibt sich im experimentellen Versuch eine mittlere Belastung $F \approx 1,1$ N auf ein frei stehendes Modellgebäude.

Khanduri setzt für seine Untersuchungen einen Windkanal mit den Ausmaßen 12 m × 1,8 m × 1,8 m ein. Für die virtuelle Simulation wird hingegen ein Strömungsgebiet mit der Länge $L \approx 1$ m und einer Höhe und Breite von $H = B \approx 0,38$ m gewählt. Dabei werden die Seitenränder mit einer Gleitrandbedingung versehen. Das Strömungsgebiet wird anschließend in der Länge mit 512 Gitterknoten und in der Höhe und Breite mit jeweils 192 Gitterknoten aufgelöst. Dabei beträgt der Abstand der Gitterknoten $\Delta x = 2$ cm.

In dieser Konstellation ergibt sich aus der Simulation für ein isoliertes Gebäude eine mittlere Windbelastung von $F \approx 0,95$ N. Diese Größe wird als Referenzwert für die Berechnung des Interferenzfaktors in Gruppenanordnung angesetzt. Im Folgenden wird der Einfluss eines zusätzlichen Gebäudes vor und hinter dem bereits existierenden Gebäude untersucht, wobei der Abstand der Gebäude (Sx) variiert wird. Der virtuelle Versuchsaufbau ist in Abbildung 7.6 veranschaulicht. Die Ergebnisse der Simulation werden in Abbildung 7.7 denen aus der Literatur [142, 159, 238, 273] gegenübergestellt. Dargestellt ist der Interferenzfaktor, jeweils bezogen auf das vordere oder hintere Gebäude in Abhängigkeit von dem jeweiligen Gebäudeabstand. Es wird ersichtlich, dass die Simulationsergebnisse speziell bei Betrachtung der Interferenzeffekte auf das hintere Gebäude qualitativ gut mit den experimentellen Daten übereinstimmen.

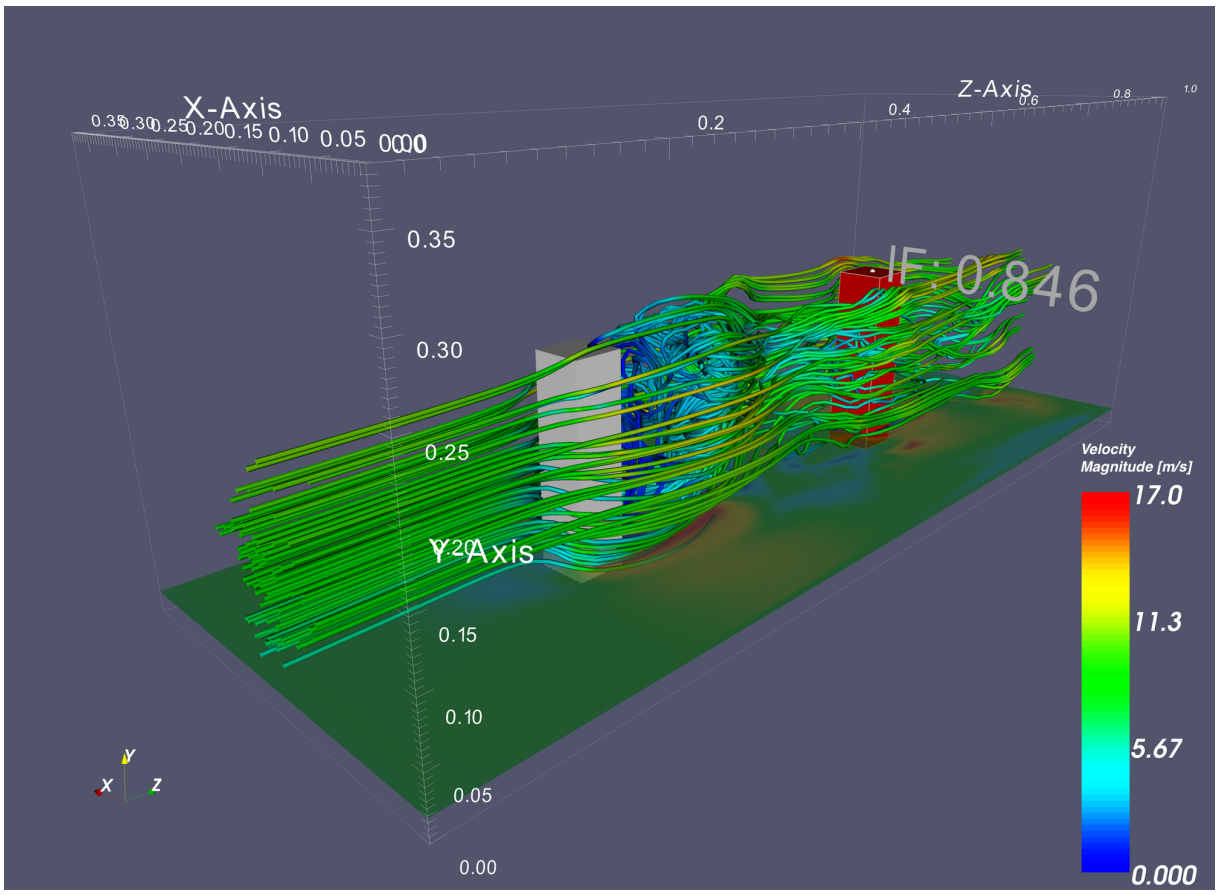
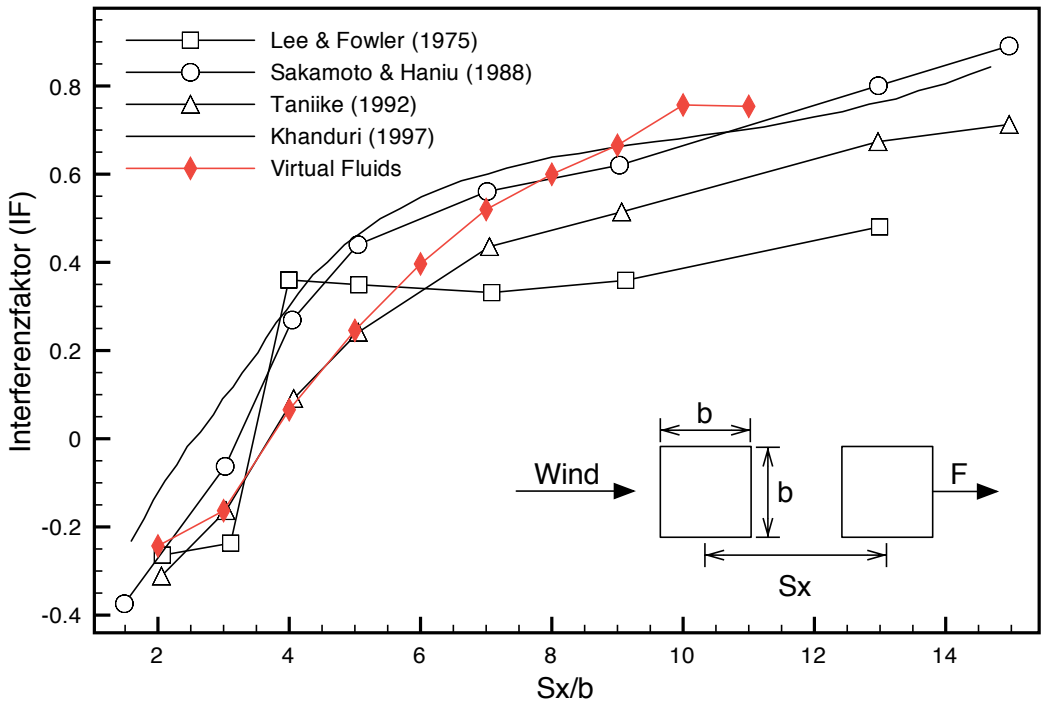
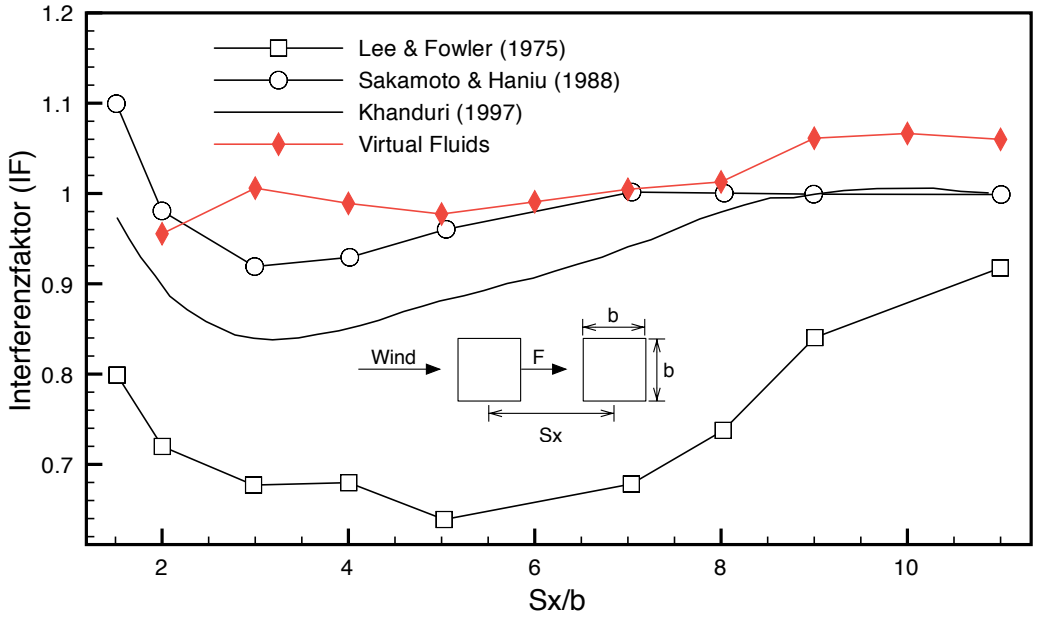


Abbildung 7.6: *Versuchsaufbau in der Simulationsumgebung. Es wird der Interferenzfaktor (IF) für das hintere Gebäude (rot dargestellt) in Abhängigkeit der Position des vorderen Gebäudes (grau dargestellt) berechnet. Es wird ersichtlich, dass in Folge des grauen Gebäudes ein Abschirmungseffekt eintritt, der zu einem Interferenzfaktor IF < 1 führt. Der Interferenzfaktor IF = 1 entspricht dem Referenzwert für das alleinstehende Gebäude.*



(a) *Interferenzfaktor bezogen auf das hintere Gebäude.*



(b) *Interferenzfaktor bezogen auf das vordere Gebäude.*

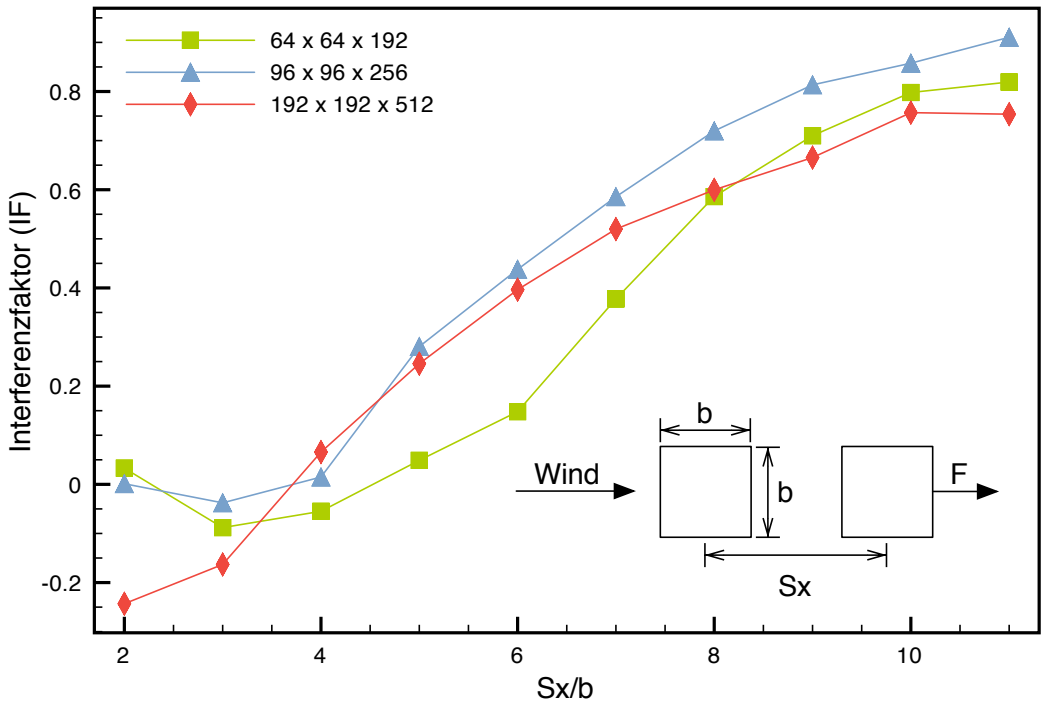
Abbildung 7.7: *Windinduzierte Interferenzen für Gebäude in Gruppenanordnung bei unterschiedlichen Gebäudeabständen.*

7.1.3 Interaktivität

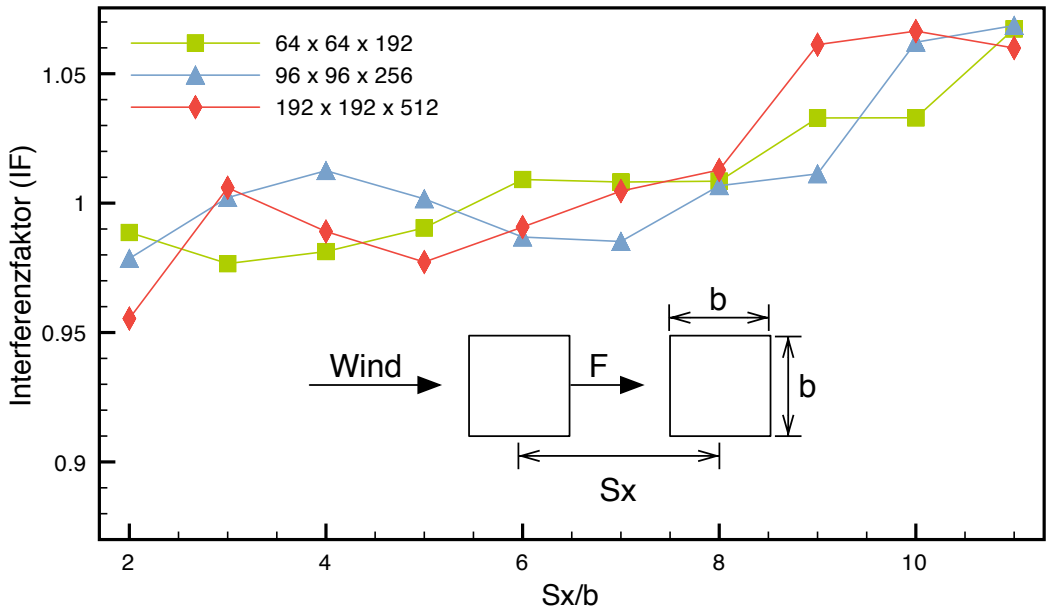
In der derzeitigen Situation ist das Verhältnis von Simulationszeit zu simulierter Zeit bezogen auf die Realität primär auf Grund der tendenziell hohen Gitterauflösung von ca. 19 Mio. Knoten für eine interaktive Simulation tendenziell ungeeignet. Bei Verwendung von vier Tesla C1060 ($\approx$ 1400 MNUPS) beträgt das Verhältnis in etwa $\frac{1}{4000}$. Demnach entspricht eine Stunde Simulationszeit etwa einer Sekunde realer Zeit. Im Folgenden wird davon ausgegangen, dass die Umlaufzeit bezogen auf das gesamte Gitter als ein Maß für die Zeit dienen kann, in der sich Änderungen an der Modellgeometrie (spätestens) in den Simulationsergebnissen widerspiegeln. Im ungünstigsten Fall muss ein Anwender demnach im Anschluss an eine Interaktion ca. 6 min warten, bis er verlässliche Ergebnisse erhält. Um die Interaktivität zu steigern, wird der Modellversuch daher mit geringeren Auflösungen wiederholt und mit dem vorherigen Ergebnis verglichen. Die Gegenüberstellung der Resultate ist Abbildung 7.8 zu entnehmen. Im Fall einer Auslösung von $64 \times 64 \times 192$ Gitterknoten und dem Einsatz von zwei Tesla C1060 ($\approx$ 700 MNUPS) reduziert sich das Verhältnis der Simulationszeit zur simulierten Zeit dabei auf $\frac{1}{100}$. Gleichzeitig wird dabei die Umlaufzeit auf eine benutzerfreundlichen Größenordnung von ca. zwölf Sekunden gesenkt.

7.1.4 Fazit

VIRTUALFLUIDS INTERACTIVE stellt ein Werkzeug zur Verfügung, mit dessen Hilfe windinduzierte Interferenzen bei Gebäuden prognostiziert werden können. Dies kann wahlweise hoch interaktiv oder alternativ mit höherer Genauigkeit erfolgen. Ein mögliches Vorgehen könnte darin bestehen, die Gebäudepositionen und Interferenzen zunächst in einer interaktiven Sitzung zu ermitteln und anschließend bei höherer Genauigkeit zu verifizieren. Im Gegensatz zur experimentellen Untersuchung fallen dabei deutlich geringere Kosten an. Verglichen mit der Ableitung von Bemessungsgrößen aus der Literatur hat der Simulationsansatz zudem den Vorteil, auch für komplexe Geometrien Anwendung finden zu können. An dem einleitenden Beispiel anknüpfend wäre somit grundsätzlich auch eine Bemessung von Kühltürmen denkbar (vgl. Abbildung 7.9).



(a) Interferenzfaktor bezogen auf das hintere Gebäude.



(c) Interferenzfaktor bezogen auf das vordere Gebäude.

Abbildung 7.8: *Mit* VIRTUALFLUIDS INTERACTIVE *ermittelte Interferenzfaktoren bei unterschiedlichen Gitterauflösungen für Gebäude in Gruppenanordnung.*

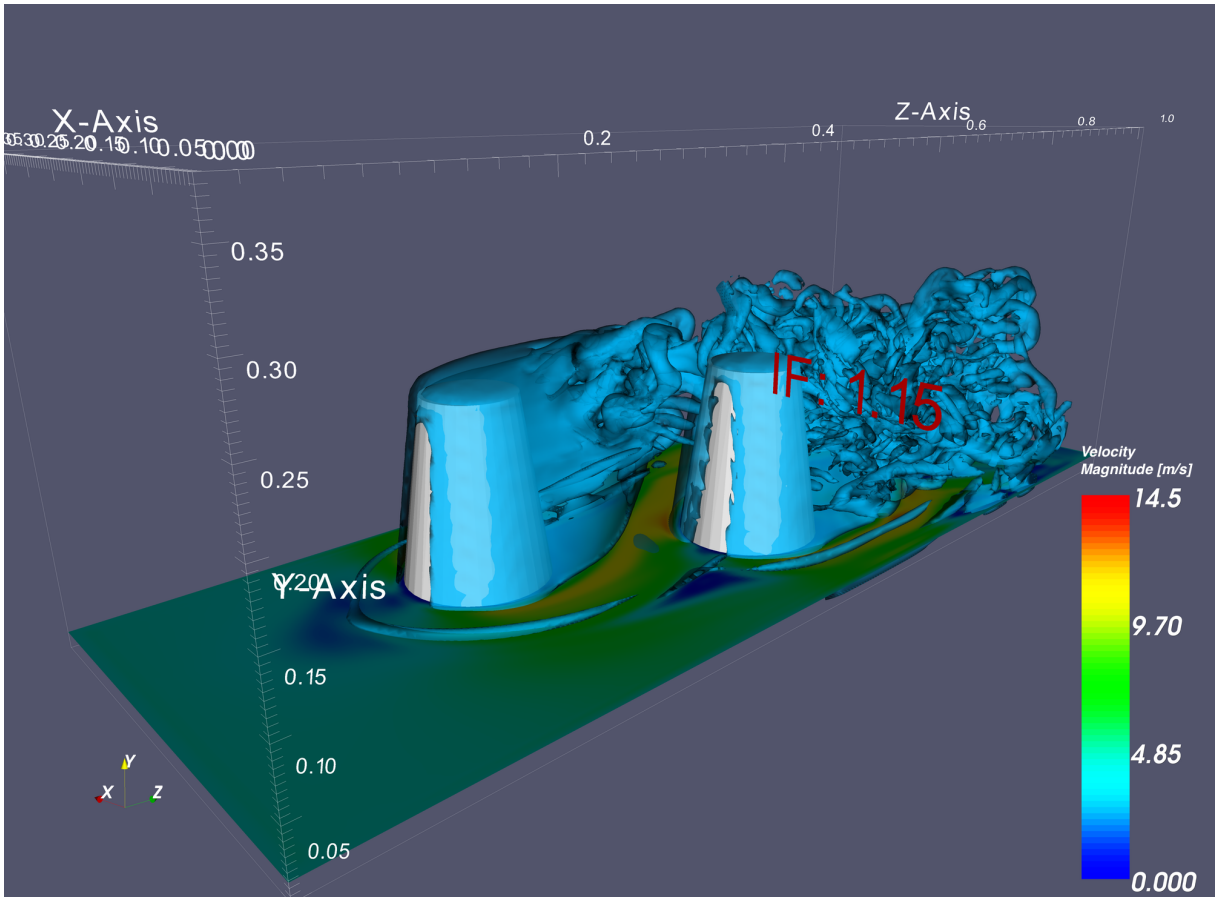


Abbildung 7.9: *Interferenzfaktorberechnung mit* VIRTUALFLUIDS INTERACTIVE *am Beispiel von Kühltürmen.*

7.2 Digitales Permeameter

Strömungsprozesse in porösen Medien spielen in vielen Bereichen der angewandten Wissenschaft und des Ingenieurwesens eine tragende Rolle. Dabei stehen zahlreiche Alltagsphänomene in enger Relation mit Transportvorgängen in unterschiedlichsten Medien, wie zum Beispiel der Transport von Wasser in Pflanzen oder die Ausbreitung von Düngemitteln oder Schadstoffen im Erdreich. Ferner sind vergleichbare Transportphänomene häufig in technisch industriellen Prozessen involviert. Ein besseres Verständnis der beteiligten Strömungsphänomene kann dabei dazu beitragen, die Prozesse hinsichtlich ihres Energiebedarfs zu optimieren. Die wohl wichtigste phänomenologische Gesetzmäßigkeit der beteiligten Transportprozesse wurde von Darcy entdeckt [242]. Sie definiert die Permeabilität als Leitfähigkeit des porösen Materials.

7.2.1 Gesetz von Darcy

Das Gesetz von Darcy, benannt nach dem französischen Ingenieur Henri Darcy, ist eine empirisch ermittelte mathematische Beschreibung laminarer Strömungen in porösen Medien:

$$q = \frac{-k}{\mu} \nabla P \tag{7.4}$$

Obwohl ursprünglich 1856 experimentell am Beispiel des Flusses durch einen homogenen Sandfilter hergeleitet, sind die Zusammenhänge von allgemeiner Gültigkeit, da gezeigt werden kann, dass Gleichung 7.4 eine spezielle Lösung der Stokes-Gleichungen darstellt. Die Darcy-Beschreibung von Stokes-Strömungen in porösen Medien ist insbesondere deshalb von großer Relevanz, da sie die strömungsrelevanten Eigenschaften eines porösen Mediums in einer einzelnen Konstante (k) [m^2] zusammenfasst. Dieser Koeffizient wird auch als hydraulische Leitfähigkeit bezeichnet und steht für die spezifische Permeabiliät. Als weitere Proportionalitätskonstante bestimmt μ [$\frac{kg}{ms}$] die dynamische Viskosität des Fluides. Der eigentliche Fluss in einem festgelegten räumlichen Intervall stellt sich in Folge des Druckgradienten ∇P [$\frac{Pa}{m}$] ein. Dabei wird q [$\frac{m}{s}$] als sogenannte Darcy-Geschwindigkeit bezeichnet, mit der die mittlere Fließgeschwindigkeit in dem homogenen Medium bezogen auf eine spezifische Flächeneinheit angegeben wird.

7.2.2 Bestimmung hydraulischer Materialeigenschaften mit LBM

Die Erkenntnisse von Darcy finden unter anderem Anwendung bei der Bestimmung der hydraulischen Eigenschaften von porösen Materialien. Ein klassischer Ansatz, diese Parameter zu ermitteln, besteht in Versuchsreihen von Ein- und Ausflussexperimenten, bei denen entsprechende Proben des porösen Mediums mit Wasser gesättigt und anschließend wieder entwässert werden [226]. Die Experimente sind mit einigen Herausforderungen verbunden. Beispielsweise ist das 100-prozentige Saturieren einer Probe mit Problemen verbunden. Eine Computersimulation kann hier als unterstützendes Hilfsmittel dienen. So setzen Ahrenholz et al. [5] ein analoges Vorgehen auf Basis der Lattice-Boltzmann-Methode ein, um die hydraulischen Eigenschaften mittels eines numerischen Ansatzes zu ermitteln.

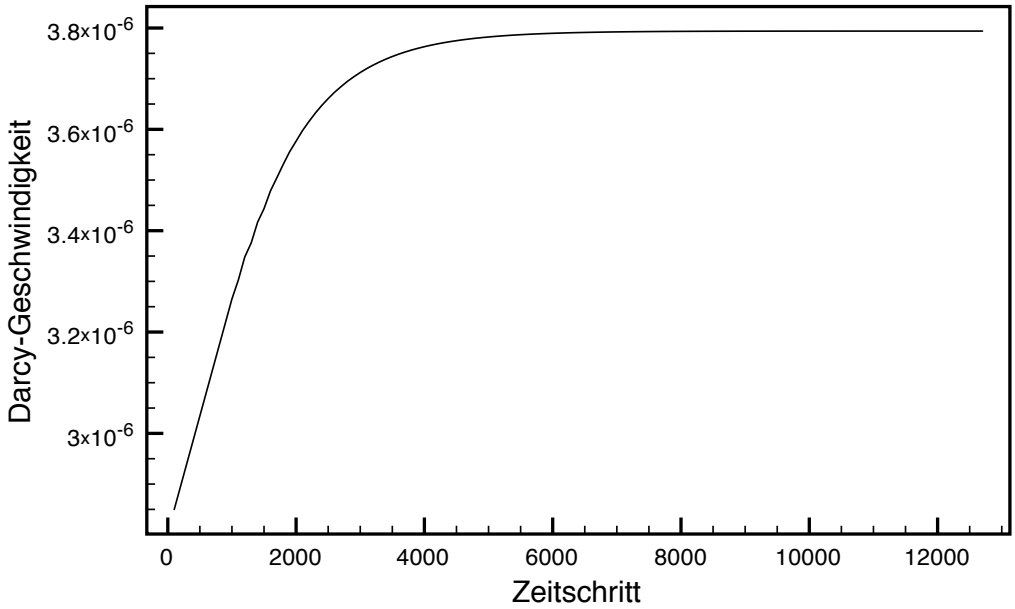


Abbildung 7.10: *Darcy-Geschwindigkeit in Abhängigkeit des Lattice-Boltzmann-Zeitschritts bei Durchströmung einer Bodenprobe mit einem Berechnungsgitter von ca. 16 Mio. Knoten.*

Die Lattice-Boltzmann-Methode eignet sich deshalb für die numerische Simulation von Strömungen in porösen Medien, weil sie naturgemäß komplexe Geometrien abbilden kann. Derartige Geometrien werden in der Regel mit Hilfe der Computertomographie-Technologie (CT) gewonnen, deren Ergebnisse leicht in eine Voxeldarstellung überführt werden können. Im Idealfall kann diese Repräsentation unmittelbar in die Simulation einfließen. Dabei wird die Auflösung der Diskretisierung auf Grund der komplexen porösen Strukturen im Allgemeinen sehr hoch sein, so dass die Systeme oft nur mit Hilfe des Einsatzes von Hochleistungsrechnern gelöst werden können. Dies gilt im Besonderen, wenn die Randbedingungen lediglich erster Ordnung genau sind (vgl. Abschnitt 5.1.2). Da die notwendigen Rechenkapazitäten häufig jedoch nicht zur Verfügung stehen, beschränken sich die Simulationen entweder auf den zweidimensionalen Anwendungsfall oder die Gitterauflösung wird alternativ zu Gunsten der Laufzeit reduziert [160, 278]. Häufig ist die Aussagekraft der Ergebnisse in diesem Fall jedoch nur von qualitativem Wert [4]. Durch den Einsatz von GPUs ist es indes möglich, auch bei höheren Auflösungen akzeptable Rechenzeiten zu erzielen.

Im Folgenden wird eine Erweiterung von VIRTUALFLUIDS INTERACTIVE zur Ermittlung der Permeabilität poröser Probekörper beschrieben. Dabei werden Geometrien verarbeitet, die bereits in einer Voxelbeschreibung, beispielsweise basierend auf CT-Daten, vorliegen. Auf dem Strömungsgebiet wird dazu ein Druckgradient aufgebracht, von dem ein Fluss induziert wird. Wenn sich nach einer gewissen Zeit eine Stationarität eingestellt hat, kann somit an Hand der mittleren Geschwindigkeit im Gebiet nach Gleichung 7.4 die Permeabilität des Mediums bestimmt werden. Abbildung 7.10 zeigt den Verlauf der Darcy-Geschwindigkeit, d.h. der mittleren Geschwindigkeit in Abhängigkeit vom aktuellen Zeitschritt, für ein Gebiet von ca. 16 Mio. Knoten. Ermittelt wird die Permeabilität einer Bodenprobe (vgl. Abbildung 7.11). Es wird ersichtlich, dass sich der stationäre Zustand und damit die gesuchte Geschwindigkeit nach spätestens 10^4 Zeitschritten einstellt. Bei einer Leistungsfähigkeit der GPU-gestützten Simulation von 700 MNUPS liegt das Ergebnis dabei in weniger als vier Minuten

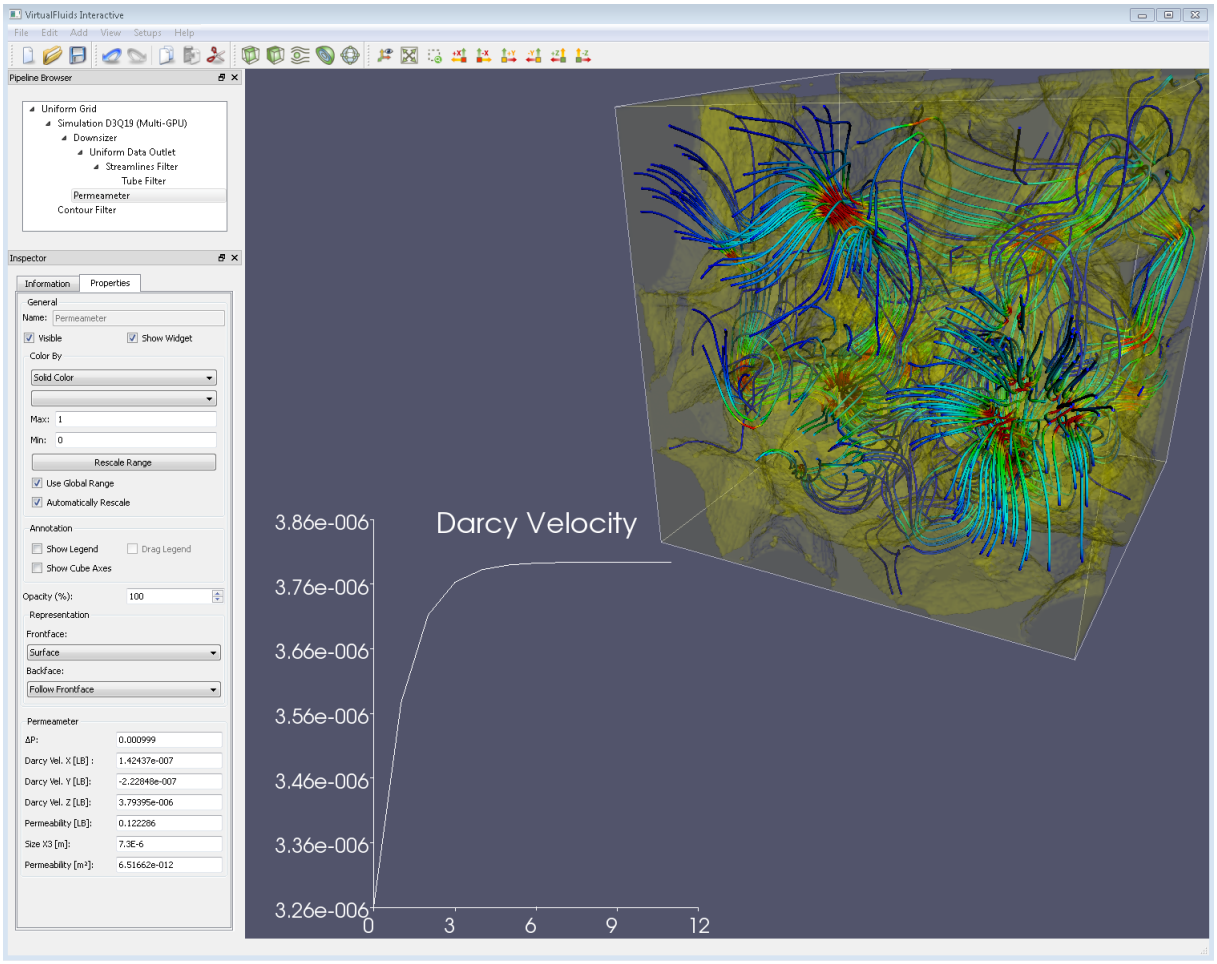


Abbildung 7.11: *Berechnung der Permeabilität einer Bodenprobe bei einer Gitterauflösung von ca. 16 Mio. Knoten in* VIRTUALFLUIDS INTERACTIVE.

vor. Die Ergebnisse der Simulation stimmen qualitativ mit den Resultaten von Ahrenholz [4] überein. Quantitativ jedoch existieren gewisse Abweichungen, die mit der Genauigkeit der Randbedingungen zu erklären sind. An dieser Stelle wäre eine Erhöhung der Auflösung notwendig, die jedoch im Widerspruch zu der gewünschten geringen Laufzeit und hohen Interaktivität steht. Allerdings ist zu erwarten, dass mit der Steigerung der Leistungsfähigkeit zukünftiger GPU-Generationen auch die Gitterauflösung erhöht werden kann. Alternativ wäre auch ein Ansatz für Randbedingungen zweiter Ordnung vielversprechend. Hinsichtlich GPU-basierter Implementierungen gibt es auf diesem Gebiet jedoch noch keine Vorarbeiten.

7.2.3 Fazit

Die vorausgehend erläuterten Anwendungsbeispiele verdeutlichen, dass VIRTUALFLUIDS INTERACTIVE trotz seines prototypischen Charakters eine solide Rahmenumgebung für interaktive GPU-basierte Strömungssimulationen für verschiedene ingenieurwissenschaftliche Fragestellungen zur Verfügung stellt. Dabei können die vorgestellten anwendungsspezifischen Erweiterungen wegweisend für zukünftige Entwicklungen sein. So erlaubt es die offene und flexible Anwendungsarchitektur, die

Simulationsumgebung auf einfache Art und Weise um zusätzliche physikalische Modelle zu erweitern und damit beispielsweise die Berechnung von Advektions- und Diffusionsproblemen [265] oder auch sogenannter freier Oberflächen zu ermöglichen. Vor diesem Hintergrund wurde exemplarisch die Arbeit von Janßen [130] integriert (vgl. Abbildung 7.12) und damit die interaktive Simulation freier Oberflächen erlaubt.

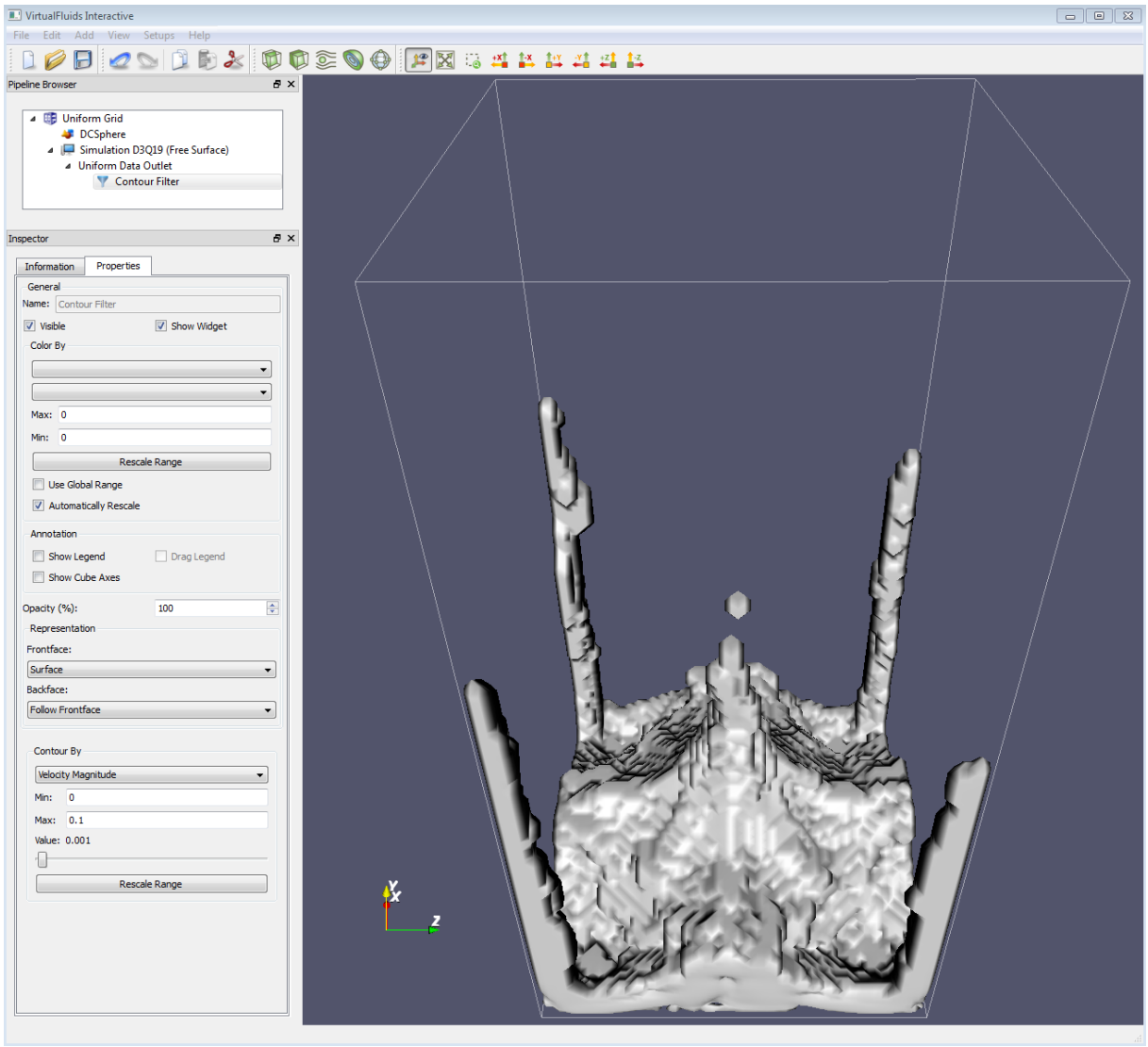


Abbildung 7.12: *GPU-basierte LB-Simulation einer freien Oberfläche nach [130].*

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Simulationen können ein wertvolles Instrument zur Lösung komplexer Aufgaben von Ingenieuren und Wissenschaftlern darstellen. Jedoch erfordert die Näherungslösung der zugrunde liegenden Gleichungen im Bereich der Strömungsmechanik den Einsatz erheblicher rechnerischer Ressourcen, die in der Regel nur von traditionellen Hochleistungsrechnern erbracht werden können. Der Einsatz derartiger Systeme ist allgemein mit erheblichen finanziellen Investitionen verbunden und erfordert darüber hinaus in der Regel ausgesprochene interdisziplinäre Fachkenntnisse seitens des Anwenders, weshalb die numerische Strömungssimulation vielen potentiellen Anwendern als Hilfsmittel verschlossen bleibt.

Im Rahmen dieser Arbeit wird hingegen ein interaktiver Simulationsprozess basierend auf dem *Computational Steering*-Paradigma motiviert, der zu einer Steigerung der Produktivität und damit zu einer Kostenreduktion beitragen kann. Gleichzeitig bietet das vorgeschlagene Verfahren die Möglichkeit, die Qualität der Benutzerschnittstelle signifikant zu steigern und damit eine intuitive und effiziente Bedienung zu ermöglichen. Weiterhin werden Gründe für die bislang geringe Verbreitung des Paradigmas diskutiert und im Wesentlichen mit den Restriktionen und der Komplexität des traditionellen Hochleistungsrechnens begründet. Demgegenüber wird in dieser Arbeit ein Ansatz verfolgt, der sich die Leistung moderner Grafikprozessoren zu Nutze macht. In diesem Vorgehen grenzt sich die Arbeit von den wenigen bereits bestehenden interaktiven Systemen ab. Verglichen mit konventionellen Ansätzen im Hochleistungsrechnen bieten GPU-basierte sowohl in der Anschaffung als auch im Betrieb einen deutlichen Kostenvorteil, so dass sie nicht nur finanzstarken Unternehmen und Bildungseinrichtungen vorbehalten bleiben. Durch einen Verbund aus mehreren GPUs stehen somit auch bei einem verhältnismäßig geringen finanziellen Aufwand ausreichend Ressourcen für Simulationen relevanter Gebietsgrößen zur Verfügung. Gleichsam ist indes eine Reduktion auf ein einzelnes Gesamtsystem möglich, wodurch sowohl Aspekte der Entwicklung als auch der externen Anwendungsqualität deutlich profitieren. Der Mehrwert lässt sich im Wesentlichen auf eine deutlich verminderte Komplexität und eine erhöhte Datenlokalität zurückführen.

Der Weg zu einer hohen externen Anwendungsqualität zu Gunsten des Anwenders führt generell über eine ausgeprägte interne Qualität. Vor diesem Hintergrund wird bei dem Anwendungsentwurf ein besonderer Wert auf die internen Qualitätsfaktoren gelegt und entsprechende Konzepte und Prinzipien der Softwareentwicklung diskutiert. Dabei wird dem Aspekt der Erweiterbarkeit eine zentrale Bedeutung beigemessen. Insofern ist VIRTUALFLUIDS INTERACTIVE als offene Rahmenanwendung konzipiert ist, die anderen Entwicklern Schnittstellen für eigene Erweiterungen bietet. Dabei abstrahiert die Anwendungsumgebung komplexe Aspekte, wie beispielsweise die Visualisierung von Ergebnisdaten und bietet dem Entwickler eine einheitliche Schnittstelle für eigene Erweiterungen. Exemplarisch werden einige bau-spezifische Erweiterungen umgesetzt und somit einerseits zur Ergebnisvalidierung dienen und andererseits die Relevanz interaktiver Strömungssimulationen für die

Ingenieurpraxis demonstrieren. Die Beispiele veranschaulichen, dass numerische Strömungssimulationen längst nicht mehr nur einigen wenigen hoch spezialisierten und finanzstarken Einrichtungen vorbehalten bleiben müssen, sondern in naher Zukunft eine hilfreiche Unterstützung des allgemeinen Ingenieuralltags darstellen können. Der »Digitale Windkanal« nach der Idee von Neumanns ist damit auf dem besten Wege, Realität zu werden.

8.2 Ausblick

Die im Rahmen dieser Arbeit entwickelte Simulationsumgebung VIRTUALFLUIDS INTERACTIVE besitzt zunächst einen prototypischen Charakter. Sie stellt jedoch eine solide Rahmenumgebung für interaktive GPU-basierte Strömungssimulationen zur Verfügung, die wegweisend für zukünftige Entwicklungen sein kann. Für einen praxisnahen Einsatz einer Simulationsumgebung ist dabei neben korrekten und flexiblen physikalischen Modellen insbesondere auch die Unterstützung komplexer Geometrien von Bedeutung. Während es VIRTUALFLUIDS INTERACTIVE grundsätzlich erlaubt, beliebig komplexe Geometrien in den Simulationsprozess einzubinden, sind die Möglichkeiten zur Modifikation des geometrischen Modells bislang begrenzt. Um Abhilfe zu schaffen, ist es denkbar, eine Anbindung an externe Applikationen wie AutoCAD [11] oder Google SketchUp [109] zu schaffen und ihre Funktionalität für Modellierungsaufgaben einzubeziehen. Änderungen an der Geometrie erfordern in der Regel eine Aktualisierung der Voxelbeschreibung. Die Aktualisierung dieser Beschreibung führt insbesondere bei großen Berechnungsgittern zu spürbaren Unterbrechungen laufender Simulationen. Obgleich an dieser Stelle bereits optimierte Algorithmen zum Einsatz kommen [23], ist ein deutlich beschleunigter Prozess bei Verwendung von GPU-basierten Algorithmen zu erwarten. Auf diese Weise würde gleichzeitig auch eine Erweiterung hinsichtlich einer Fluid-Struktur-Interaktion denkbar, die eine kontinuierliche Veränderung der Geometrie mit sich bringt. Da die derzeitige Implementierung für jeden diskreten Knoten des Gebiets (unabhängig vom Knotentyp) einen vollständigen Verteilungssatz vorsieht, bleibt im Fall großer ortsfester Strukturen unter Umständen ein erheblicher Speicheranteil ungenutzt. Dies gilt insbesondere für poröse Medien wie in Abschnitt 7.2 behandelt. Um dem vorzubeugen, kann beispielsweise ein Speicherschema verwendet werden, das eine indirekte Adressierung verwendet [280]. Zusätzlich ist es sinnvoll, die Auflösung in bestimmten Bereichen des Strömungsgebiets zu erhöhen. Dahingehend existieren am iRMB bereits Ansätze zur Gitterverfeinerung auf GPUs, die in VIRTUALFLUIDS INTERACTIVE einfließen können [243]. Gleichzeitig ist es notwendig, flexible Randbedingungen zu realisieren, die es erlauben zusätzliche Ein- und Ausflussränder festzulegen. Bislang beschränken sich Ein- und Auslassrandbedingungen auf die äußeren Ränder der Strömungsgebiets. Des Weiteren wäre es sinnvoll, die Funktionalität der Anwendung dahingehend zu erweitern, dass Ergebnisse reproduzierbar berechnet und einzelne Zustände zwischengespeichert werden können (*Checkpointing*). Zusätzlich wäre eine weitere Annäherung von Visualisierung und Simulation denkbar. So könnte die Responsivität der Anwendung in erheblichem Maße davon profitieren, dass Teile der Visualisierung zusammen mit der Simulation auf der GPU ausgeführt werden und im Idealfall direkt zur Anzeige gebracht werden können. Drüber hinaus kann die Unterstützung von OpenCL [110] zu einer Steigerung der Anwenderfreundlichkeit beitragen, da der Einsatzbereich von VIRTUALFLUIDS INTERACTIVE durch die Verwendung von CUDA bislang auf NVIDIA-GPUs begrenzt ist. Durch eine alternative Implementierung unter Verwendung von OpenCL ist es jedoch möglich, diese Einschränkung auszuräumen. Auf Grund der hohen Interaktivität der

Anwendung eignet sich VIRTUALFLUIDS INTERACTIVE derzeit besonders für das *Rapid Prototyping* im Vorfeld von Produktionsläufen. Diese praxisrelevanten Simulationen zeichnen sich im Allgemeinen durch eine immense Systemgröße aus, weshalb die Berechnung auf einem einzelnen System zumeist nicht möglich ist. Alternativ können die Simulationen auf Cluster-Systemen durchgeführt werden, deren einzelne Knoten mit GPUs ausgestattet sind. Zur Unterstützung der Benutzer im Umgang mit derartigen Cluster-Systemen wäre es denkbar, VIRTUALFLUIDS INTERACTIVE für eine parallele Berechnung zu erweitern. Obwohl die oft langandauernden Produktionsläufe meist wenig Benutzerinteraktion erfordern, kann die Simulationsumgebung dabei als komfortable Benutzerschnittstelle zu dem Hochleistungsrechner dienen und den Anwender beispielsweise beim Aufsetzen der Simulation und der Kontrolle laufender Simulationen unterstützen. Da die Simulationsumgebung plattformunabhängig entworfen wurde, ist es möglich, die Anwendung direkt auf einem Cluster-Knoten auszuführen und im Fernzugriff zu bedienen. Auf diese Weise wird es vermieden, die Ergebnisdatenmenge für die Analyse zu dem Arbeitsplatzrechner des Anwenders zu transferieren. Zusätzlich erschließt sich die Simulationsumgebung gleichzeitig einer größeren Anzahl Nutzern und ermöglicht zudem ein kollaboratives Arbeiten. Der Aspekt der Zusammenarbeit konnte bereits erfolgreich sowohl auf einem Linux-basierten als auch auf einem Windows-basierten Cluster getestet werden.

Literaturverzeichnis

[1] Abelson, H., Sussman, G. J. und with Julie Sussman: *Structure and Interpretation of Computer Programs*. MIT Press/McGraw-Hill, Cambridge, 1. Aufl., 1985.

[2] Abram, G. und Treinish, L.: *An extended data-flow architecture for data analysis and visualization*. SIGGRAPH Comput. Graph., 29(2):17–21, 1995.

[3] Advanced Visual Systems: *AVS/Express*. `http://www.avs.com/software/soft_t/avsxps.html`, besucht am 19. Februar 2011.

[4] Ahrenholz, B.: *Massively parallel simulations of multiphase- and multicomponent flows using lattice Boltzmann methods*. Dissertation, Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2009.

[5] Ahrenholz, B., Tölke, J., Lehmann, P., Peters, A., Kaestner, A., Krafczyk, M. und Durner, W.: *Prediction of capillary hysteresis in a porous material using lattice-Boltzmann methods and comparison to experimental data and a morphological pore network model*. Advances in Water Resources, 31(9):1151 – 1173, 2008.

[6] Alexander, C.: *The Timeless Way of Building*. Oxford University Press, New York, 1979.

[7] Anderson, J. R.: *Cognitive Psychology and its Implications*. W. H. Freeman and Co., New York, NY, 1980.

[8] ANSYS: *ANSYS FLUENT Flow Modeling Software*. `http://www.ansys.com/products/fluid-dynamics/fluent/`, besucht am 19. Februar 2011.

[9] ANSYS: *ANSYS CFX*, 2010. `http://www.ansys.com/products/fluid-dynamics/cfx/`, besucht am 19. Februar 2011.

[10] Armitt, J.: *Wind Loading on Cooling Towers*. Journal of the Structural Division, 106(3):623–641, 1980.

[11] Autodesk: *AutoCAD*, 2010. `http://autodesk.com/`, besucht am 4. Oktober 2010.

[12] Avila, L., Barre, S., Blue, R., Cole, D., Geveci, B., Hoffman, W. A., King, B., Law, C. C., Martin, K. M., Schroeder, W. J. und Squillacote, A. H.: *The VTK User's Guide*. Kitware Inc., 2006.

[13] Axner, L.: *High performance computational hemodynamics with the Lattice Boltzmann method*. Dissertation, Universiteit van Amsterdam, 2007.

[14] Bailey, A. und Vincent, N. D. G.: *Wind-Pressure on buildings including effects of adjacent buildings*. Journal of the ICE, 20:243–275, 1943.

[15] Bailey, P., Myre, J., Walsh, S., Lilja, D. und Saar, M.: *Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors*. In: *Parallel Processing, 2009. ICPP '09. International Conference on*, S. 550 –557, 2009.

[16] Becker, M. und Wössner, U.: *Tangible interfaces for interactive flow simulation.* In: Krause, E., Shokin, Y., Resch, M. und Shokina, N. (Hrsg.): *Computational Science and High Performance Computing II*, Bd. 91 d. Reihe *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, S. 253–260. Springer Berlin / Heidelberg, 2006.

[17] Belleman, R. G.: *Interactive Exploration in Virtual Environments.* Dissertation, Universiteit van Amsterdam, April 2003.

[18] Bellemann, R. G. und Shulakov, R.: *High Performance Distributed Simulation for Interactive Simulated Vascular Reconstruction.* In: *ICCS '02: Proceedings of the International Conference on Computational Science-Part III*, S. 265–274, London, UK, 2002. Springer-Verlag.

[19] Benzi, R., Succi, S. und Vergassola, M.: *The lattice Boltzmann equation: theory and applications.* Physics Reports, 222(3):145–197, 1992.

[20] Bertin, J.: *Graphics and graphic information-processing.* Walter de Gruyter, 1981.

[21] Bertin, J.: *Semiology of graphics: diagrams, networks, maps.* University of Wisconsin Press, 1983.

[22] Bhatnagar, P. L., Gross, E. P. und Krook, M.: *A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems.* Phys. Rev., 94(3):511–525, May 1954.

[23] Bindick, S.: *Ein integrierter Ansatz zur interaktiven dreidimensionalen Simulation gekoppelter thermischer Prozesse.* Dissertation, Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2010.

[24] Blake, R. J., Coveney, P. V., Clarke, P. und Pickles, S. M.: *The TeraGyroid experiment – Supercomputing 2003.* Sci. Program., 13(1):1–17, 2005.

[25] Blelloch, G. E.: *Prefix Sums and Their Applications.* In: *Synthesis of Parallel Algorthms.* Morgan Kaufmann Publishers, 1990.

[26] Blelloch, G. E.: *Vector models for data-parallel computing.* MIT Press, Cambridge, MA, USA, 1990.

[27] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G. und Merritt, M.: *Characteristics of Software Quality.* Elsevier Science, 1978.

[28] Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Connallen, J. und Houston, K. A.: *Object-oriented analysis and design with applications.* Addison Wesley, 3. Aufl., 2007.

[29] Borrmann, A.: *Computerunterstützung verteiltkooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken.* Dissertation, Technische Universität München, 2007.

[30] Borrmann, A., Wenisch, P., Treeck, C. van und Rank, E.: *Collaborative Computational Steering: Principles and Application in HVAC Layout.* Integrated Computer-Aided Engineering, 13(4):361–376, 2006.

[31] Bower, A. und McGlashan, B.: *Twisting the Triad: The evolution of the Dolphin Smalltalk MVP application framework.* In: *European Smalltalk User Group (ESUG)*, 2000.

[32] Brodlie, K., Brooke, J., Chen, M., Chisnall, D., Fewings, A. J., Hughes, C., John, N. W., Jones, M. W., Riding, M. und Roard, N.: *Visual Supercomputing: Technologies, Applications and Challenges*. Comput. Graph. Forum, 24(2):217–245, 2005.

[33] Brodlie, K., Duce, D., Gallop, J., Sagar, M., Walton, J. und Wood, J.: *Visualization in Grid Computing Environments*. In: *VIS '04: Proceedings of the conference on Visualization '04*, S. 155–162, Washington, DC, USA, 2004. IEEE Computer Society.

[34] Brodlie, K., Duce, D., Gallop, J. und Wood, J.: *Distributed Cooperative Visualization: State of the Art Report*. In: *Proceedings of Eurographics '98*, 1998.

[35] Brodlie, K., Poon, A., Wright, H., Brankin, L., Banecki, G. und Gay, A.: *GRASPARC: a problem solving environment integrating computation and visualization*. In: *Proceedings of the 4th conference on Visualization '93*, S. 102–109, Washington, DC, USA, 1993. IEEE Computer Society.

[36] Brodlie, K. und Wood, J.: *Computational Steering in Visualization Dataflow Environments*. In: Oxley, L. und Kulasiri, D. (Hrsg.): *MODSIM 2007 International Congress on Modelling and Simulation*, S. 3077–3083, 2007.

[37] Brodlie, K., Wood, J., Duce, D. und Sagar, M.: *gViz - Visualization and Computational Steering on the Grid*. In: Cox, S. J. (Hrsg.): *Proceedings of the UK e-Science All Hands Meeting 2004*, S. 54–60, 2004.

[38] Brodlie, K. W., Carpenter, L., Earnshaw, R. A., Gallop, J. R., Hubbold, R. J., Mumford, A. M., Osland, C. D. und Quarendon, P. (Hrsg.): *Scientific visualization: techniques and applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[39] Brooke, J. M., Coveney, P. V., Harting, J., Jha, S., Pickles, S. M., Pinning, R. L. und Porter, A. R.: *Computational Steering in RealityGrid*. In: *UK e-Science All Hands Meeting*, 2003.

[40] Brooks, F. P.: *Grasping reality through illusion—interactive graphics serving science*. In: *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*, S. 1–11, New York, NY, USA, 1988. ACM.

[41] Brooks, Jr., F. P.: *No Silver Bullet Essence and Accidents of Software Engineering*. Computer, 20(4):10–19, 1987.

[42] Bryson, S.: *The Virtual Windtunnel: A High-Performance Virtual Reality Application*. In: *VR*, S. 20–26, 1993.

[43] Bryson, S.: *Virtual Environments in Scientific Visualization*. In: *In Virtual Reality for Visualization, Course Notes of Tutorial 5 at Visualization 95*. Course, 1995.

[44] Bryson, S.: *Virtual reality in scientific visualization*. Communucations of the ACM, 39(5):62–71, 1996.

[45] Bryson, S.: *Direct Manipulation in Virtual Reality*. In: Johnson, C. und Hansen, C. (Hrsg.): *The Visualization Handbook*. Academic Press, Inc., Orlando, FL, USA, 2004.

[46] Bryson, S. und Gerald-Yamasaki, M.: *The distributed virtual windtunnel*. In: *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, S. 275–284, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[47] Bryson, S. und Levit, C.: *The virtual wind tunnel.* Computer Graphics and Applications, IEEE, 12(4):25 –34, jul 1992.

[48] Buck, E. M. und Yacktman, D. A.: *Cocoa Design Patterns.* Addison-Wesley Professional, 2009.

[49] Budgen, D.: *Software Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[50] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. und Stal, M.: *Pattern-Oriented Software Architecture: a system of patterns*, Bd. 1. John Wiley and Sons, 1996.

[51] Buyya, R.: *High Performance Cluster Computing: Architectures and Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[52] Cameron, G.: *Modular visualization environments: past, present, and future.* SIGGRAPH Comput. Graph., 29(2):3–4, 1995.

[53] Card, S. K., Mackinlay, J. D. und Shneiderman, B. (Hrsg.): *Readings in information visualization: using vision to think.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[54] Chapman, S. und Cowling, T. G.: *The Mathematical Theory of Non-uniform Gases.* Cambridge University Press, 3. Aufl., 1991.

[55] Chen, J., Rine, D. und Simon, H.: *Advancing Interactive Visualization and Computational Steering.* Computational Science Engineering, IEEE, 3(4):13 –17, 1996.

[56] Chen, S. und Doolen, G. D.: *Lattice Boltzmann method for fluid flows.* Annual Review of Fluid Mechanics, 30(1):329–364, 1998.

[57] Chernoff, H.: *The Use of Faces to Represent Points in K-Dimensional Space Graphically.* Journal of the American Statistical Association, 68(342):361–368, 1973.

[58] Chin, J., Harting, J., Jha, S., Coveney, P. V., Porter, A. R. und Pickles, S. M.: *Steering in computational science: mesoscale modelling and simulation.* Contemporary Physics, S. 417–434, 2003.

[59] Chmilar, M. und Wyvill, B.: *A software architecture for integrated modeling and animation.* In: Earnshaw, R. A. und Wyvill, B. (Hrsg.): *New Advances in Computer Graphics: Proceedings of CG International '89*, S. 257–276. Springer-Verlag, 1989.

[60] Chu, N. S. H. und Tai, C. L.: *Real-time ink dispersion in absorbent paper.* ACM Transactions on Graphics, 24:504–511, 2005.

[61] Chung, T. J.: *Computational Fluid Dynamics.* Cambridge University Press, 2. Aufl., 2010.

[62] Coltheart, M.: *The Persistences of Vision.* Philosophical Transactions of the Royal Society of London. B, Biological Sciences, 290:57–69, 1980.

[63] Conner, B. D., Snibbe, S. S., Herndon, K. P., Robbins, D. C., Zeleznik, R. C. und Dam, A. van: *Three-dimensional widgets.* In: *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, S. 183–188, New York, NY, USA, 1992. ACM.

[64] Coren, S., Ward, L. und Enns, J.: *Sensation and Perception.* J. Wiley & Sons, 2004.

[65] Crane, K., Llamas, I. und Tariq, S.: *Real-Time Simulation and Rendering of 3D Fluids.* In: Nguyen, H. (Hrsg.): *GPU Gems 3*, Kap. 30. Addison Wesley Professional, August 2007.

[66] Crouse, B.: *Lattice-Boltzmann Strömungssimulationen auf Baumdatenstrukturen.* Dissertation, Technische Universität München, 2003.

[67] Crow, F. C.: *The aliasing problem in computer-generated shaded images.* Commun. ACM, 20:799–805, 1977.

[68] Dam, A. van, Forsberg, A. S., Laidlaw, D. H., LaViola, J. J. und Simpson, R. M.: *Immersive VR for Scientific Visualization: A Progress Report.* IEEE Computer Graphics and Applications, 20:26–52, 2000.

[69] d'Auriac, J. C. A., Barthou, D., Becirevic, D., Bilhaut, R., Bodin, F., Boucaud, P., Brand-Foissac, O., Carbonell, J., Eisenbeis, C., Gallard, P., Grosdidier, G., Guichon, P., Honoré, P. F., Meur, G. L., Pène, O., Rilling, L., Roudeau, P., Seznec, A., Stocchi, A. und Touze, F.: *Towards the petaflop for Lattice QCD simulations the PetaQCD project.* Journal of Physics: Conference Series, 219(5), 2010.

[70] DeMarco, T.: *Structured Analysis and System Specification.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1979.

[71] Deutsches Institut für Normung e. V.: *DIN EN 1991-1-4: Eurocode 1: Einwirkungen auf Tragwerke - Teil 1-4: Allgemeine Einwirkungen - Windlasten*, 2005.

[72] d'Humières, D., Bouzidi, M. und Lallemand, P.: *Thirteen-velocity three-dimensional lattice Boltzmann model.* Phys. Rev. E, 63(6):066702, May 2001.

[73] d'Humières, D., Ginzburg, I., Krafczyk, M., Lallemand, P. und Luo, L. S.: *Multiple-relaxation-time lattice Boltzmann models in three dimensions.* Royal Society of London Philosophical Transactions Series A, 360:437–451, 2002.

[74] Diehl, S.: *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software.* Springer, 2007.

[75] Dijkstra, E. W.: *The humble programmer.* Commun. ACM, 15(10):859–866, 1972.

[76] Dijkstra, E. W.: *Programming considered as a human activity.* In: Yourdon, E. N. (Hrsg.): *Classics in software engineering*, S. 1–9. Yourdon Press, Upper Saddle River, NJ, USA, 1979.

[77] Dijkstra, E. W.: *On the role of scientific thought (EWD447).* In: *Selected Writings on Computing: A Personal Perspective*, S. 60–66. Springer-Verlag, 1982. First published: Neuen, The Netherlands, August 1974.

[78] Doleisch, H.: *SIMVIS: interactive visual analysis of large and time-dependent 3D simulation data.* In: *WSC '07: Proceedings of the 39th conference on Winter simulation*, S. 712–720, Piscataway, NJ, USA, 2007. IEEE Press.

[79] Ebert, D. S.: *Extending Visualization to Perceptualization: The Importance of Perception in Effective Communication of Information.* In: Johnson, C. und Hansen, C. (Hrsg.): *The Visualization Handbook*. Academic Press, Inc., Orlando, FL, USA, 2004.

[80] Eickermann, T. und Frings, W.: *VISIT — a Visualization Interface Toolkit Version 1.0.* Internal Report IB-2000, Forschungszentrum Jülich, 2000.

[81] Esnard, A., Richart, N. und Coulaud, O.: *A Steering Environment for Online Parallel Visualization of Legacy Parallel Simulations.* In: *DS-RT '06: Proceedings of the 10th IEEE international*

symposium on Distributed Simulation and Real-Time Applications, S. 7–14, Washington, DC, USA, 2006. IEEE Computer Society.

[82] Euclid: *The Elements*. St. John's College Press, 1947.

[83] Exa: *PowerFLOW*. `http://www.exa.com/pages/pflow/pflow_main.html`, besucht am 19. Februar 2011.

[84] Fahrig, T.: *Kooperative Optimierung von Raumluftströmungen mittels agentengestützter Regelungstechnik in einer Computational Steering Umgebung*. Dissertation, Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2007.

[85] Fan, Z., Qiu, F., Kaufman, A. und Yoakum-Stover, S.: *GPU Cluster for High Performance Computing*. In: *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, S. 47. IEEE Computer Society, 2004.

[86] Fatica, M.: *Accelerating linpack with CUDA on heterogenous clusters*. In: *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, S. 46–51, New York, NY, USA, 2009. ACM.

[87] Feichtinger, C., Habich, J., Köstler, H., Hager, G., Rüde, U. und Wellein, G.: *A Flexible Patch-Based Lattice Boltzmann Parallelization Approach for Heterogeneous GPU-CPU Clusters*. Parallel Computing, In Press, Accepted Manuscript, 2011.

[88] Ferziger, J. und Perić, M.: *Computational methods for fluid dynamics*. Springer, 2002.

[89] Flynn, M. J.: *Some Computer Organizations and Their Effectiveness*. IEEE Transactions on Computers, C-21(9):948 –960, 1972.

[90] Foster, I. und Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[91] Fowler, M.: *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[92] Fowler, M.: *Development of Further Patterns of Enterprise Application Architecture*. "Website", 2010. Available online at `http://www.martinfowler.com/eaaDev/index.html`, visited on 2010-02-15.

[93] Freeman, S. und Pryce, N.: *Growing Object-Oriented Software, Guided by Tests*. Addison-Wesley Professional, 2009.

[94] Freudiger, S.: *Entwicklung eines parallelen, adaptiven, komponentenbasierten Strömungskerns für hierarchische Gitter auf Basis des Lattice-Boltzmann-Verfahrens*. Dissertation, Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2009.

[95] Frisch, U., d'Humières, D., Hasslacher, B., Lallemand, P., Pomeau, Y. und Rivet, J. P.: *Lattice Gas Hydrodynamics in Two and Three Dimensions*. Complex Systems, 1:75–136, 1987.

[96] Funkhouser, T. A. und Séquin, C. H.: *Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments*. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, S. 247–254, New York, NY, USA, 1993. ACM.

[97] Gamma, E., Helm, R., Johnson, R. E. und Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[98] Gannon, J. D., Hamlet, R. G. und Mills, H. D.: *Theory of Modules*. IEEE Trans. Softw. Eng., 13(7):820–829, 1987.

[99] Geier, M.: *Ab initio derivation of the cascaded lattice Boltzmann automaton*. Dissertation, Department of Microsystems Technology IMTEK,University of Freiburg, 2006.

[100] Geier, M., Greiner, A. und Korvink, J. G.: *Cascaded digital lattice Boltzmann automata for high Reynolds number flow*. Physical Revue E, 73(6), 2006.

[101] Geier, M., Greiner, A. und Korvink, J. G.: *Bubble functions for the lattice Boltzmann method and their application to grid refinement*. The European Physical Journal - Special Topics, 171:173–179, 2009.

[102] Geier, M., Linxweiler, J. und Krafczyk, M.: *EsoStripe*. zur Veröffentlichung vorgesehen, 2011.

[103] Geist, G. A., II, Arthur, J., Kohl, J. A. und Papadopoulos, P. M.: *CUMULVS: Providing Fault-Tolerance, Visualization And Steering Of Parallel Applications*. International Journal of High Performance Computing Applications, 11:224–236, 1996.

[104] Geller, S.: *Ein explizites Modell für die Fluid-Struktur-Interaktion basierend auf LBM und p-FEM*. Dissertation, Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2010.

[105] Ginzburg, I. und d'Humières, D.: *Local second-order boundary methods for lattice Boltzmann models*. Journal of Statistical Physics, 84:927–971, 1996.

[106] Glass, R. L.: *Software Creativity*. Addison Wesley Professional, 1995.

[107] Globus, A.: *A Software Model for Visualization of Large Unsteady 3-D CFD Results*. In: *AIAA 95-0115, 33rd Aerospace Sciences Meeting and Exhibit, January 912*, S. 92–031, 1991.

[108] Goldberg, A. und Robson, D.: *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[109] Google Inc.: *Google SketchUp*, 2011. `http://sketchup.google.com/`, besucht am 30. Mai 2011.

[110] Group, K.: *OpenCL - The open standard for parallel programming of heterogeneous systems*. `http://www.khronos.org/opencl/`, besucht am 22. Februar 2011.

[111] Haber, R. und McNabb, D.: *Visualization idioms: a conceptual model for scientific visualization systems*. In: Nielson, G., Shriver, B. und Rosenblum, L. (Hrsg.): *Visualization in scientific computing*, S. 74–93. IEEE Computer Society Press, Silver Spring, 1990.

[112] Haber, R. B., McNabb, D. A. und Ellis, R. A.: *Eliminating distance in scientific computing: an experiment in televisualization*. Int. J. Supercomput. Appl. High Perform. Eng., 4(4):71–89, 1990, ISSN 0890-2720.

[113] Hager, G. und Wellein, G.: *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Inc., Boca Raton, FL, USA, 2010.

[114] Haines, R., McKeown, M., Pickles, S., Pinning, R., Porter, A. und Riding, M.: *The service architecture of the TeraGyroid experiment*. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 363(1833):1743–1755, 2005.

[115] Hansen, C., Johnson, C. R., Pascucci, V. und Silva, C. T.: *Visualization for Data-Intensive Science.* In: Hey, T., Tansley, S. und Tolle, K. (Hrsg.): *The Fourth Paradigm: Data-Intensive Scientific Discovery*, S. 153–164. Microsoft Research, Redmond, Washington, 2009.

[116] Harlow, F. H. und Welch, J. E.: *Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface.* Physics of Fluids, 8(12):2182–2189, 1965.

[117] Harris, M., Sengupta, S. und Owens, J. D.: *Parallel Prefix Sum (Scan) with CUDA.* In: Nguyen, H. (Hrsg.): *GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Kap. 39, S. 851–876. Addison Wesley, 2007.

[118] Hart, D. und Kraemer, E.: *Consistency Considerations in the Interactive Steering of Computations.* International Journal of Parallel and Distributed Systems and Networks, 2:171–179, 1999.

[119] He, X. und Luo, L. S.: *Theory of the lattice Boltzmann method: from the Boltzmann equation to the lattice Boltzmann equation.* Physical Revue E, 56:6811–6817, 1997.

[120] Higuera, F. J. und Jiménez, J.: *Boltzmann Approach to Lattice Gas Simulations.* Europhysics Letters, 9(7), 1989.

[121] Higuera, F. J., Succi, S. und Benzi, R.: *Lattice Gas Dynamics with Enhanced Collisions.* Europhysics Letters, 9(4), 1989.

[122] Hillegass, A.: *Cocoa Programming for Mac OS X.* Addison-Wesley Longman, 3. Aufl., 2008.

[123] Hoffman, D. D.: *Visual intelligence: How we create what we see.* W.W. Norton amp; Co., 2000.

[124] Holmes, I. R. und Kawalsky, R. S.: *The RealityGrid PDA and Smartphone clients: Developing effective handheld user interfaces for e-Science.* In: Cox, S. J. (Hrsg.): *Proceedings of UK e-Science All Hands Meeting*, S. 502–509. EPSRC, 2006.

[125] Hou, S., Sterling, J., Chen, S. und Doolen, G. D.: *A Lattice Boltzmann Subgrid Model for High Reynolds Number Flows.* Fields Institute Communications, S. 151–166, 1994.

[126] Houstis, E., Gallopoulos, E., Bramley, R. und Rice, J.: *Problem-Solving Environments for Computational Science.* Computing in Science and Engineering, 4:18–21, 1997.

[127] Hunt, A. und Thomas, D.: *The Pragmatic Programmer: From Journeyman to Master.* Addison-Wesley Professional, 1999.

[128] Ingalls, D. H. H.: *The Smalltalk-76 programming system design and implementation.* In: *POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, S. 9–16, New York, NY, USA, 1978. ACM.

[129] Ingalls, D. H. H.: *Design Principles Behind Smalltalk.* Byte, 6(8):286–298, 1981.

[130] Janßen, C. und Krafczyk, M.: *Free surface flow simulations on GPGPUs using the LBM.* Computers Mathematics with Applications, 2011.

[131] Johnson, C.: *Top scientific visualization research problems.* Computer Graphics and Applications, IEEE, 24(4):13 – 17, 2004.

[132] Johnson, C. und Hansen, C.: *The Visualization Handbook.* Academic Press, Inc., Orlando, FL, USA, 2004.

[133] Johnson, C., Parker, S. G., Hansen, C., Kindlmann, G. L. und Livnat, Y.: *Interactive Simulation and Visualization.* Computer, 32(12):59–65, 1999.

[134] Johnson, C. R. und Parker, S. G.: *A computational steering model applied to problems in medicine.* In: *Supercomputing '94: Proceedings of the 1994 conference on Supercomputing*, S. 540–549, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[135] Johnson, C. R., Ross, R., Ahern, S., Ahrens, J., Bethel, W., Ma, K. L., Papka, M., Rosendale, H. W. S. John van und Thomas, J.: *Visualization and Knowledge Discovery: Report from the DOE/ASCR Workshop on Visual Analysis and Data Exploration at Extreme Scale*, 2007.

[136] Johnson, R. W.: *Handbook of Fluid Dynamics.* CRC Press, 1998.

[137] Jung, V.: *Integrierte Benutzerunterstützung für die Visualisierung in Geo-Informationssystemen.* Fraunhofer IRB Verlag, 1998.

[138] Karlsson, B.: *Beyond the C++ standard library : an introduction to Boost.* Addison-Wesley, 2006.

[139] Kay, A.: *Email Message Sent to the Squeak Mailing List.* http://lists.squeakfoundation.org/pipermail/squeak-dev/1998-October/017019.html, 1998.

[140] Keefe, D.: *Integrating Visualization and Interaction Research to Improve Scientific Workflows.* Computer Graphics and Applications, IEEE, 30(2):8 –13, march-april 2010.

[141] Keim, D. A., Müller, W. und Schumann, H.: *Information Visualization and Visual Data Mining.* In: *Proceedings Eurographics 2002*, Saarbrücken, 2002.

[142] Khanduri, A. C.: *Wind-induced interference effects on buildings : integrating experimental and computerized approaches.* Dissertation, Concordia University, 1997.

[143] Kirk, D. B. und Hwu, W. m. W.: *Programming Massively Parallel Processors: A Hands-on Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1. Aufl., 2010.

[144] Kitware: *ParaView.* `http://www.paraview.org/`, besucht am 28. Mai 2011.

[145] Kitware: *The Visualization Toolkit (VTK).* `http://www.vtk.org/`, besucht am 13. September 2010.

[146] Kok, A. J. F. und Liere, R. van: *A multimodal virtual reality interface for 3D interaction with VTK.* Knowl. Inf. Syst., 13(2):197–219, 2007.

[147] Kolmogorov, A. N.: *Dissipation of Energy in the Locally Isotropic Turbulence.* Comptes rendus (Doklady) de l'Académie des Sciences de l'U.R.S.S., XXXII(1), 1941.

[148] Krafczyk, M.: *Simulation von Strömungen mit Gittergasmethoden.* Dissertation, Universität Dortmund, 1995.

[149] Krafczyk, M.: *Die Gitter-Boltzmann-Methode: Von der Theorie zur Awendung.* Habilitationsschrift, Technische Universität München, 2001.

[150] Krafczyk, M., Tölke, J. und Fahrig, T.: *Ein Prototyp für verteilte, interaktiv-kooperative Simulationen zur Beschleunigung von Entwurfszyklen im Konstruktiven Ingenieurbau.* In: Rüppel, U. (Hrsg.): *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau*, S. 273–293. Springer Berlin Heidelberg, 2007.

[151] Krafczyk, M., Tölke, J. und Luo, L. S.: *Large-Eddy Simulations with a Multiple-Relaxation LBE Model*. International Journal of Modern Physics B, 17:33–39, 2003.

[152] Kreylos, O., Bethel, E. W., Ligocki, T. J. und Hamann, B.: *Virtual-Reality Based Interactive Exploration of Multiresolution Data*. In: *In Hierarchical and Geometrical Methods in Scientific Visualization*, S. 205–224. Springer Verlag, 2003.

[153] Kreylos, O., Tesdall, A. M., Hamann, B., Hunter, J. K. und Joy, K. I.: *Interactive visualization and steering of CFD simulations*. In: *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, S. 25–34. Eurographics Association, 2002.

[154] Kühner, S.: *Virtual Reality-basierte Analyse und interaktive Steuerung von Strömungssimulationen im Bauwesen*. Dissertation, Technische Universität München, 2003.

[155] Kundu, P. K. und Cohen, I. M.: *Fluid Mechanics, Fourth Edition*. Academic Press, 2007.

[156] Lane, D. A.: *Visualization of time-dependent flow fields*. In: *VIS '93: Proceedings of the 4th conference on Visualization '93*, S. 32–38, Washington, DC, USA, 1993. IEEE Computer Society.

[157] Larmann, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall PTR, 2002.

[158] Larus, J. und Gannon, D.: *Multicore Computing and Scientific Discovery*. In: Hey, T., Tansley, S. und Tolle, K. (Hrsg.): *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[159] Lee, B. E.: *The effect of turbulence on the surface pressure field of a square prism*. Journal of Fluid Mechanics, 69(02):263–282, 1975.

[160] Lehmann, P., Krafczyk, M., Gygi, A., Flisch, A., Wyss, P. und Flühler, H.: *Modelling flow of water and air in reconstructed structures of porous media*. In: *Proceedings of the 2nd world congress on industrial tomography*, S. 628–635, 2001.

[161] Li, W., Wei, X. und Kaufman, A.: *Implementing lattice Boltzmann computation on graphics hardware*. The Visual Computer, 19:444–456, 2003.

[162] Lieberherr, K. J. und Holland, I.: *Assuring good style for object-oriented programs*. IEEE Software, S. 38–48, September 1989.

[163] Liere, R. V. und Wijk, J. J. V.: *CSE - A Modular Architecture for Computational Steering*. In: *Proceedings of the 7th Eurographics Workshop on Visualization in Scientific Computing*, S. 257–266. Springer Verlag, 1996.

[164] Liere, R. van und Mulder, J. D.: *Ubiquitous Computational Steering*. In: *IEEE Visualization '97*, 1997.

[165] Liere, R. van, Mulder, J. D. und Wijk, J. J. van: *Computational steering*. Future Gener. Comput. Syst., 12(5):441–450, 1997.

[166] Linxweiler, J., Geller, S. und Zimmermann, J.: *Ein Prototyp zur immersiven Betrachtung und interaktiven Manipulation räumlicher Objekte*. In: *Forum Bauinformatik*, 2005.

[167] Linxweiler, J., Tölke, J. und Krafczyk, M.: *Applying Modern Soft- and Hardware Technologies for Computational Steering Approaches in Computational Fluid Dynamics*. In: *Cyberworlds, 2007. CW '07. International Conference on*, S. 41–45, Oct. 2007.

[168] Liskov, B.: *Data Abstraction and Hierarchy.* ACM SIGPLAN Notices, 23(5):17–34, März 1987.

[169] Lorensen, W. E. und Cline, H. E.: *Marching cubes: A high resolution 3D surface construction algorithm.* In: Stone, M. C. (Hrsg.): *SIGGRAPH*, S. 163–169. ACM, 1987.

[170] Luksch, P., Rathmayer, S. und Sunderam, V.: *Internet-based Collaborative Simulation in Computational Prototyping and Scientific Research.* In: *The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, Las Vegas, Nevada, USA, 2000.

[171] Lyman, P. und Varian, H. R.: *How much information?* The Journal of Electronic Publishing, 6(2), 2000.

[172] Ma, K. L.: *Visualizing visualizations. User interfaces for managing and exploring scientific visualization data.* Computer Graphics and Applications, IEEE, 20(5):16 –19, sep. 2000.

[173] Ma, K. L.: *In Situ Visualization at Extreme Scale: Challenges and Opportunities.* Computer Graphics and Applications, IEEE, 29(6):14 –19, nov.-dec. 2009.

[174] MacEachren, A. M.: *How Maps Work: Representation, Visualization, and Design.* The Guilford Press, New York, 2. Aufl., 2004.

[175] Mackinlay, J.: *Automating the design of graphical presentations of relational information.* ACM Trans. Graph., 5(2):110–141, 1986.

[176] Marshall, R., Kempf, J., Dyer, S. und Yen, C. C.: *Visualization methods and simulation steering for a 3D turbulence model of Lake Erie.* SIGGRAPH Comput. Graph., 24(2):89–97, 1990.

[177] Martin, R. C.: *The Dependency Inversion Principle.* The C++ Report, August 1996.

[178] Martin, R. C.: *The interface segregation principle: one of the many principles of OOP.* The C++ Report, August 1996.

[179] Martin, R. C.: *Design Principles and Design Patterns.* Object Mentor, 2000.

[180] Martin, R. C.: *Agile Software Development: Principles, Patterns, and Practices.* Prentice Hall, 2002.

[181] Martin, R. C., Feathers, M. C., Ottinger, T. R., Langr, J. J., Grenning, B. L. S. J. W. und Wampler, K. D.: *Clean Code: A Handbook of Agile Software Craftsmanship.* Robert C. Martin Series. Prentice Hall, Upper Saddle River, NJ, 2008.

[182] Martin, R. C. und Martin, M.: *Agile Principles, Patterns, and Practices in C.* Robert C. Martin Series. Prentice Hall, Upper Saddle River, NJ, 2006.

[183] Martin, R. C., Riehle, D. und Buschmann, F.: *Pattern Languages of Program Design 3.* Addison-Wesley, 1997.

[184] Mattila, K., Hyväluoma, J., Rossi, T., Aspnäs, M. und Westerholm, J.: *An efficient swap algorithm for the lattice Boltzmann method.* Computer Physics Communications, 176(3):200 – 210, 2007.

[185] Mattila, K., Hyväluoma, J., Timonen, J. und Rossi, T.: *Comparison of implementations of the lattice-Boltzmann method.* Computers Mathematics with Applications, 55(7):1514 – 1524, 2008.

[186] McCall, J.: *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*, Bd. 1-3. General Electric, November 1977.

[187] McConnell, S.: *Code Complete, Second Edition.* Microsoft Press, Redmond, WA, USA, 2004.

[188] McCormick, B., Fanti, T. D. und Brown, M.: *Visualization in Scientific Computing: Report of the NSF Advisory Panel on Graphics, Image Processing and Workstations.* ACM Computer Graphics, 21(6), 1987.

[189] McLoughlin, T., Laramee, R. S., Peikert, R., Post, F. H. und Chen, M.: *Over Two Decades of Integration-Based, Geometric Flow Visualization.* Computer Graphics Forum, 2010.

[190] McNamara, G. R. und Zanetti, G.: *Use of the Boltzmann Equation to Simulate Lattice-Gas Automata.* Physical Review Letters, 61(20):2332–2335, 1988.

[191] Meyer, B.: *Object-Oriented Software Construction (2nd Edition).* Prentice Hall PTR, 2. Aufl., March 2000.

[192] Miller, G. A.: *The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.* The Psychological Review, 63(2):81–97, March 1956.

[193] Mittelstaedt: *Entwicklung eines objektorientierten, VTK-basierten 3D-VirtualReality- Strömungssimulators für das Lattice-Boltzmann-Verfahren.* Diplomarbeit, 2004.

[194] Moore, G. E.: *Cramming More Components onto Integrated Circuits.* Electronics, 1965.

[195] Mulder, J. D., Wijk, J. J. van und Liere, R. van: *A survey of computational steering environments.* Future Gener. Comput. Syst., 15(1):119–129, 1999.

[196] Mulder, J. D. und Wijk, J. van: *3D Computational Steering with Parametrized Geometric Objects.* In: *Visualization '95 (Proceedings of the 1995 Visualization Conference*, S. 304–311. Press, 1995.

[197] Munzner, T., Johnson, C., Moorhead, R., Pfister, H., Rheingans, P. und Yoo, T. S.: *NIH-NSF Visualization Research Challenges Report Summary.* IEEE Computer Graphics and Applications, 26:20–24, 2006.

[198] NASA - National Aeronautics and Space Administration: *Space Shuttle RTF Ascent Aerodynamics and Debris Transport Analysis.* `http://www.hec.nasa.gov/news/gallery_images/gomez.shuttle.html`, besucht am 12. September 2010.

[199] Neumann, J. v.: *First draft of a report on the EDVAC.* Techn. Ber., University of Pennsylvania, 1945.

[200] Neumann, J. von: *The Principles of Large-Scale Computing Machines.* Annals of the History of Computing, 10(4):243 –256, 1988. Reprint.

[201] Nguyen, N. Q. und Ladd, A. J. C.: *Sedimentation of hard-sphere suspensions at low Reynolds number.* Journal of Fluid Mechanics, 525:73–104, 2005.

[202] Nielson, G. M., Hagen, H. und Müller, H.: *Scientific Visualization: Overviews, Methodologies, and Techniques.* IEEE-CS Press, Los Alamitos, CA, USA, 1997.

[203] Norfolk, M.: *Knottingley and Ferrybridge online*, 2011. `www.knottingley.org`, besucht am 22. Mai 2011.

[204] Norman, D. A.: *The psychology of everyday things.* Doubleday, 1988.

[205] Norman, D. A.: *Things that make us smart: defending human attributes in the age of the machine.* Addison-Wesley, 1994.

[206] NVIDIA: *CUDA Zone – The resource for CUDA developers.* "Website", 2010. `http://www.nvidia.com/object/cuda_home_new.html`, besucht am 16. Februar 2011.

[207] NVIDIA: *NVIDIA CUDA C Best Practices Guide.* NVIDIA, 3.2 Aufl., 2010.

[208] NVIDIA: *NVIDIA CUDA C Programming Guide.* NVIDIA, 3.2 Aufl., 2010.

[209] NVIDIA: *Tesla C2050/C2070 GPU Computing Processor - Supercomputing at 1/10th the Cost*, 2010.

[210] NVIDIA: *NVIDIA CUDA C Programming Guide.* NVIDIA, 4.0 Aufl., 2011. Release Candidate 2.

[211] Oden, J. T., Fish, J., Johnson, C., Laub, A., Srolovitz, D., Belytschko, T., Hughes, T. J., Keyes, D., Petzold, L. und Yip, S.: *Report of the NSF Blue Ribbon Panel on Simulation-Based Engineering Science*, 2006.

[212] Padua, D. A. und Hoeflinger, J. P.: *Supercomputers.* In: *Encyclopedia of Computer Science*, S. 1710–1718. John Wiley and Sons Ltd., Chichester, UK, 2003.

[213] Paivio, A.: *Mental Representations: A Dual Coding Approach.* Oxford University Press, 1990.

[214] Parker, S. G., Hansen, C. D., Johnson, C. R. und Miller, M.: *Computational Steering and the SCIRun Integrated Problem Solving Environment.* In: *DAGSTUHL '97: Proceedings of the Conference on Scientific Visualization*, S. 257, Washington, DC, USA, 1997. IEEE Computer Society.

[215] Parker, S. G. und Johnson, C. R.: *SCIRun: a scientific programming environment for computational steering.* In: *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, S. 52, New York, NY, USA, 1995. ACM.

[216] Parker, S. G., Johnson, C. R. und Beazley, D.: *Computational Steering Software Systems and Strategies.* IEEE Comput. Sci. Eng., 4(4):50–59, 1997.

[217] Parker, S. G., Weinstein, D. W. und Johnson, C. R.: *The SCIRun computational steering software system.* In: Arge, E., Bruaset, A. M. und Langtangen, H. P. (Hrsg.): *Modern software tools for scientific computing*, S. 5–44. Birkhauser Boston Inc., Cambridge, MA, USA, 1997.

[218] Parnas, D. L.: *On the Criteria To Be Used in Decomposing Systems into Modules.* Communications of the ACM, 15(12):1053–1058, December 1972.

[219] Parnas, D. L.: *A Technique for Software Module Specification with Examples.* Commun. ACM, 15(5):330–336, 1972.

[220] Parnas, D. L.: *Software aspects of strategic defense systems.* Commun. ACM, 28(12):1326–1335, 1985.

[221] Peter, L. J.: *The Peter Pyramid.* William Morrow & Company, 1986.

[222] Pickles, S. M., Haines, R., Pinning, R. L. und Porter, A. R.: *A practical toolkit for computational steering.* Philosophical Transactions: Mathematical, Physical and Engineering Sciences, 363(1833):1843–1853, 2005.

[223] Playfair, W.: *The commercial and political Atlas.* London, UK, 1785.

[224] Pope, S.: *Turbulent Flows.* Cambridge University Press, Cambridge, 2000.

[225] Potel, M.: *MVP: Model-View-Presenter, The Taligent Programming Model for C++ and Java*. Techn. Ber., Taligent, 1996.

[226] Priesack, E. und Durner, W.: *Closed-Form Expression for the Multi-Modal Unsaturated Conductivity Function*. Vadose Zone Journal, 5(1):121–124, 2006.

[227] Qian, Y. H., d'Humières, D. und Lallemand, P.: *Lattice BGK models for Navier-Stokes equation*. Europhysics Letters, 17(6):479–484, 1992.

[228] Qiu, F., Zhao, Y., Fan, Z., Wei, X., Lorenz, H., Wang, J., Yoakum-Stover, S., Kaufman, A. und Mueller, K.: *Dispersion simulation and visualization for urban security*. In: *Visualization, 2004. IEEE*, S. 553 – 560, 2004.

[229] Rantzau, D. und Lang, U.: *A scalable virtual environment for large scale scientific data analysis*. Future Gener. Comput. Syst., 14(3-4):215–222, 1998.

[230] Rantzau, D. und Thomas, P.: *Parallel cfd-simulations in a distributed high performance software environment using european atm networks*. In: *Proceedings Parallel CFD 96 Conference*, 1996.

[231] Redmond, K. C. und Smith, T. M.: *From Whirlwind to MITRE: The R&D Story of The SAGE Air Defense Computer*. MIT Press, Cambridge, 2000.

[232] Reenskaug, T.: *Models - Views - Controllers*. Techn. Ber., Technical Note, Xerox Parc, 1979.

[233] Reeves, J. W.: *Code as Design: Three Essays*. Developer, 2005. »What Is Software Design?« originally published in C++ Journal, 1992.

[234] Riel, A. J.: *Object-Oriented Design Heuristics*. Addison-Wesley, Reading, MA, 1996.

[235] Robertson, G., Card, S. K. und Mackinlay, J. D.: *The cognitive coprocessor architecture for interactive user interfaces*. In: *Proceedings of the 2nd annual ACM SIGGRAPH symposium on User interface software and technology*, UIST '89, S. 10–18, New York, NY, USA, 1989. ACM.

[236] Rothman, D. H. und Zaleski, S.: *Lattice-Gas Cellular Automata: Simple Models of Complex Hydrodynamics*. Cambridge University Press, 1997.

[237] Ruprecht, A., Eisinger, R. und Göde, E.: *Innovative Design Environments for Hydro Turbine Components*. In: *HYDRO 2000*, Oktober 2000.

[238] Sakamoto, H., Hainu, H. und Obata, Y.: *Fluctuating forces acting on two square prisms in a tandem arrangement*. Journal of Wind Engineering and Industrial Aerodynamics, 26(1):85 – 103, 1987.

[239] Sanders, J. und Kandrot, E.: *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1. Aufl., 2010.

[240] Santos, S. dos und Brodlie, K.: *Gaining understanding of multivariate and multidimensional data through visualization*. Computers & Graphics, 28(3):311 – 325, 2004.

[241] Scaife, M. und Rogers, Y.: *External cognition: how do graphical representations work?* International Journal of Human-Computer Studies, 45(2):185 – 213, 1996.

[242] Scheidegger, A. E.: *The Physics of Flow Through Porous Media*. Soil Science, 86(6), 1958.

[243] Schönherr, M., Kucher, K., Geier, M., Stiebler, M., Freudiger, S. und Krafczyk, M.: *Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs*. Computers & Mathematics with Applications, 2011. zur Veröffentlichung akzeptiert.

[244] Schroeder, W., Martin, K. und Lorensen, B.: *The Visualization Toolkit, Third Edition.* Kitware Inc., 2007.

[245] Schroeder, W. J., Avila, L. S. und Hoffman, W.: *Visualizing with VTK: A Tutorial.* IEEE Computer Graphics and Applications, 20:20–27, 2000.

[246] Schroeder, W. J., Martin, K. M. und Lorensen, W. E.: *The Design and Implementation of an Object-Oriented Toolkit for 3D Graphics and Visualization.* In: *In IEEE Visualization*, S. 93–100, 1996.

[247] Schumann, H. und Müller, W.: *Visualisierung - Grundlagen und allgemeine Methoden.* Springer, 2000.

[248] Scientific Computing and Imaging Institute at the University of Utah - SCI: *Three-Dimensional Annotation for Volume Rendering.* `http://www.sci.utah.edu/research/visualization.html?start=6`, besucht am 12. September 2010.

[249] Sells, C. und Griffiths, I.: *Programming WPF.* O'Reilly Media, 2. Aufl., 2007.

[250] Sells, C. und Weinhardt, M.: *Windows Forms 2.0 Programming.* Addison-Wesley Professional, 2. Aufl., 2006.

[251] Sheridan, T. B. und Ferrell, W. R.: *Man-machine systems: Information, control, and decision models of human performance.* MIT Press, 1974.

[252] Shneiderman, B.: *Direct manipulation: A step beyond programming languages.* S. 461–467, 1987.

[253] Shneiderman, B.: *The eyes have it: a task by data type taxonomy for information visualizations.* In: *Visual Languages, 1996. Proceedings., IEEE Symposium on*, S. 336 –343, 3-6 1996.

[254] Shneiderman, B. und Plaisant, C.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Pearson Addison-Wesley, Upper Saddle River, NJ, 5. Aufl., 2009.

[255] Simon, H.: *The Sciences of the Artificial.* MIT Press, Cambridge, 1969.

[256] Smith, J. E. und Pleszkun, A. R.: *Implementation of precise interrupts in pipelined processors.* SIGARCH Comput. Archit. News, 13:36–44, June 1985.

[257] Smith, M. H., Dalsem, W. R. van, Dougherty, R. C. und Buning, P. G.: *Analysis and Visualization of Complex Unsteady Three-Dimensional Flows.* In: *27th AIAA Aerospace Sciences Meeting and Exhibit*, 1989.

[258] Sommerville, I.: *Software Engineering.* Addison-Wesley, Harlow, England, 9. Aufl., 2010.

[259] Song, D., Golin, E. und Norman, M.: *A Fine-Grain Dataflow Model For Scientific Visualization Systems.* In: *In Workshop Papers of the Fourth Eurographics Workshop on Visualization in Scientific Computing, Abingdon, United Kingdom*, S. 21–23, 1993.

[260] Sperry, R. W.: *Hemisphere disconnection and unity in conscious awareness.* American Psychologist,, 1968.

[261] Stallings, W.: *Computer Organization and Architecture: Designing for Performance.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 7. Aufl., 2005.

[262] Stathopoulos, T.: *Introduction to Wind Engineering, Wind Structure, Wind-Building Interaction.* In: *Wind Effects on Buildings and Design of Wind-Sensitive Structures.* Springer-Verlag New York, Inc., 2007.

[263] Sterling, J. D. und Chen, S.: *Stability analysis of lattice Boltzmann methods.* Journal of Computational Physics, 123(1):196–206, 1996.

[264] Stevens, W. P., Myers, G. J. und Constantine, L. L.: *Structured Design.* IBM Systems Journal, 13(2):115–139, 1974.

[265] Stiebler, M., Tölke, J. und Krafczyk, M.: *Advection-diffusion lattice Boltzmann scheme for hierarchical grids.* Comput. Math. Appl., 55:1576–1584, 2008.

[266] Stroustrup, B.: *The C++ Programming Language.* Addison Wesley, Reading, Massachusetts, 2. Aufl., 1991.

[267] Succi, S.: *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond (Numerical Mathematics and Scientific Computation).* Oxford University Press, USA, August 2001.

[268] Summerfield, M.: *Advanced Qt programming : creating great software with C++ and Qt 4.* Addison-Wesley Professional, 2010.

[269] Sutter, H. und Larus, J.: *Software and the Concurrency Revolution.* Queue, 3(7):54–62, 2005.

[270] Tamura, Y.: *Wind Induced damage to Buildings and Disaster Risk Reduction.* In: *Proceedings of the seventh Asia Pasific Conference on Wind Engineering*, 2009.

[271] Tanenbaum, A. S.: *Computer Networks.* PH PTR, 4. Aufl., 2003.

[272] Tanenbaum, A. S.: *Modern Operating Systems.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3. Aufl., 2007.

[273] Taniike, Y.: *Interference mechanism for enhanced wind forces on neighboring tall buildings.* Journal of Wind Engineering and Industrial Aerodynamics, 42(1-3):1073 – 1083, 1992.

[274] Taylor, V., Chen, J., Disz, T., Papka, M. und Stevens, R.: *Interactive virtual reality in simulations: exploring lag time.* Computational Science Engineering, IEEE, 3(4):46 –54, 1996.

[275] The HDF Group: *HDF5 User's Guide*, 2011. Release 1.8.7.

[276] Tichy, W. F.: *The Evidence for Design Patterns.* In: Oram, A. und Wilson, G. (Hrsg.): *Making Software: What Really Works, and Why We Believe It*, Kap. 22, S. 393–413. O'Reilly Media, 2010.

[277] Tölke, J.: *Die Lattice–Boltzmann Methode für Mehrphasenstömungen.* Dissertation, Technische Universität München, 2001.

[278] Tölke, J.: *Lattice Boltzmann simulations of binary fluid flow through porous media.* Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 360(1792):535–545, 2002.

[279] Tölke, J.: *Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA.* Computing and Visualization in Science, 2008.

[280] Tölke, J.: *Lattice Boltzmann multi-phase simulations using GPUs.* GPU Technology Conference, 2010.

[281] Tölke, J. und Krafczyk, M.: *TeraFLOP computing on a desktop PC with GPUs for 3D CFD*. International Journal of Computational Fluid Dynamics, 22(7):443–456, August 2008.

[282] Tölke, J. und Krafczyk, M.: *Second order interpolation of the flow field in the lattice Boltzmann method*. Computers Mathematics with Applications, 58(5):898 – 902, 2009.

[283] Treinish, L.: *Visualization of stratospheric ozone depletion and the polar vortex*. In: *IEEE Conference on Visualization*, 1993.

[284] Tufte, E. R.: *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[285] Tufte, E. R.: *Envisioning Information*. Graphics Press, 1990.

[286] Tullsen, D. M., Eggers, S. J. und Levy, H. M.: *Simultaneous multithreading: maximizing on-chip parallelism*. In: *25 years of the international symposia on Computer architecture (selected papers)*, ISCA '98, S. 533–544, New York, NY, USA, 1998. ACM.

[287] Uffelen, G. M. van: *Wind-induced building interference: increase of wind loads on existing buildings after erection of new high-rises*. In: *In prceedings of the 5th European African Conference on Wind Engineering*, 2009.

[288] Uffelen, G. M. van: *Windinduzierte Interferenzen bei Gebäuden: Erhöhung der Windlasten auf bestehende Gebäude duch die Errichtung neuer Gebäude*. In: Peil, U. (Hrsg.): *Windingenieurwesen in Forschung und Praxis*. Windtechnologische Gesellschaft e.V., 2009.

[289] Upson, C.: *Future Directions of Visualization Software Environments*. In: *SIGGRAPH '91 panel*, 1991.

[290] Upson, C., Faulhaber, Jr., T., Kamins, D., Laidlaw, D. H., Schlegel, D., Vroom, J., Gurwitz, R. und Dam, A. van: *The Application Visualization System: A Computational Environment for Scientific Visualization*. IEEE Comput. Graph. Appl., 9(4):30–42, 1989.

[291] Vetter, J. und Schwan, K.: *Models for Computational Steering*. In: *ICCDS '96: Proceedings of the 3rd International Conference on Configurable Distributed Systems*, S. 100, Washington, DC, USA, 1996. IEEE Computer Society.

[292] Vetter, J. S. und Schwan, K.: *High Performance Computational Steering of Physical Simulations*. In: *IPPS '97: Proceedings of the 11th International Symposium on Parallel Processing*, S. 128, Washington, DC, USA, 1997. IEEE Computer Society.

[293] Walker, R., Kenny, P. und Miao, J.: *Exploratory simulation for astrophysics*. In: Erbacher, R. F., Roberts, J. C., Grohn, M. T. und Borner, K. (Hrsg.): *Visualization and Data Analysis 2007*, 2007.

[294] Walton, J.: *NAG's IRIS Explorer*. In: *Visualization Handbook*. Academic Press, Inc., Orlando, FL, USA, 2004.

[295] Wei, X., Zhao, Y., Fan, Z., Li, W., Qiu, F., Yoakum-Stover, S. und Kaufman, A. E.: *Lattice-Based Flow Field Modeling*. IEEE Transactions on Visualization and Computer Graphics, 10:719–729, 2004.

[296] Weiß, G. und Jakob, R.: *Agentenorientierte Softwareentwicklung: Methoden und Tools*. Springer, Berlin, 2004.

[297] Wellein, G. und Hager, G.: *The Lattice Boltzmann Method: Basic Performance Characteristics and Performance Modeling*. SIAM Conference on Computational Science and Engineering, 2011.

[298] Wellein, G., Zeiser, T., Hager, G. und Donath, S.: *On the single processor performance of simple lattice Boltzmann kernels*. Computers and Fluids, 35(8-9):910 – 919, 2006. Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.

[299] Welzel, B., Ruprecht, A. und Lein, G.: *Numerical Optimization of an Axial Hydraulic Turbine*. In: *NAFEMS World Congress'97*, 1997.

[300] Wenisch, P.: *Computational Steering of CFD Simulations on Teraflop-Supercomputers*. Dissertation, Technische Universität München, 2008.

[301] Wenisch, P., Wenisch, O. und Rank, E.: *Harnessing High-Performance Computers for Computational Steering*. Recent Advances in Parallel Virtual Machine and Message Passing Interface, S. 536–543, 2005.

[302] Wierse, A.: *Performance of the collaborative visualization environment (COVISE) visualization system under different conditions*. In: Grinstein, G. G. und Erbacher, R. F. (Hrsg.): *Proceedings of the SPIE '95 Conference on Visual Exploration and Analysis*, 1995.

[303] Wierse, A., Lang, U. und Rühle, R.: *Architectures of Distributed Visualization Systems and their Enhancements*. In: *Proceedings of the 4th Eurographics Workshop on Visualization in Scientific Computing*, 1993.

[304] Wijk, J. J. V., Liere, R. V., Mulder, J. D., , Wijk, J. J. V., Liere, R. V. und Mulder, J. D.: *Bringing Computational Steering to the User*. In: *Presented at the Dagstuhl Seminar on Scientific Visualization*, S. 304–313, June 1997.

[305] Wijk, J. V. und Liere, R. V.: *An Environment for Computational Steering*. In: *Centre for Mathematics and Computer Science (CWI*, S. 23–27. Computer Society Press, 1997.

[306] Wijk, J. van: *The value of visualization*. In: *Visualization, 2005. VIS 05. IEEE*, S. 79 – 86, 23-28 2005.

[307] Wirth, N.: *Program development by stepwise refinement*. Commun. ACM, 14(4):221–227, 1971.

[308] Wolf-Gladrow, D. A.: *Lattice-Gas Cellular Automata and Lattice Boltzmann Models*. Springer Verlag, 2000.

[309] Wood, J., Brodlie, K. und Walton, J.: *gViz - Visualization and Steering for the Grid*. In: *Proceedings of e-Science All Hands Meeting*, Nottingham 2003.

[310] Wood, J., Brodlie, K. und Wright, H.: *Visualization over the World Wide Web and its application to environmental data*. In: *Proceedings of the 7th conference on Visualization '96*, VIS '96, S. 81–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[311] Wood, J. D. und Wright, H.: *Steering via the image in local, distributed and collaborative settings*. Concurr. Comput. : Pract. Exper., 20(3):265–276, 2008.

[312] Wright, H.: *Putting Visualization First in Computational Steering*. In: *Proceedings of UK e-Science All Hands Meeting*, S. pp 326 – 331, August 2004.

[313] Wright, H.: *Introduction to Scientific Visualization*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[314] Wright, H., Brodlie, K. und Brown, M.: *The dataflow visualization pipeline as a problem solving environment*. In: *Proceedings of the Eurographics workshop on Virtual environments and scientific visualization '96*, S. 267–276, London, UK, 1996. Springer-Verlag.

[315] Wright, H., Crompton, R., Kharche, S. und Wenisch, P.: *Steering and visualization: Enabling technologies for computational science*. Future Generation Computer Systems, 26(3):506 – 513, 2010.

[316] Wulf, W., London, R. und Shaw, M.: *Abstraction and verification in Alphard: Introduction to language and methodology*. In: Shaw, M. (Hrsg.): *ALPHARD: Form and Content*. Springer-Verlag, New York, 1981.

[317] Yourdon, E. und Constantine, L. L.: *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press computing series. Prentice-Hall, Inc, Upper Saddle River, NJ, USA, 1979.

[318] Yu, H., Wang, C., Grout, R., Chen, J. und Ma, K. L.: *In Situ Visualization for Large-Scale Combustion Simulations*. Computer Graphics and Applications, IEEE, 30(3):45 –57, 2010.

[319] Zhao, Y., Han, Y., Fan, Z., Qiu, F., Kuo, Y. C., Kaufman, A. E. und Mueller, K.: *Visual Simulation of Heat Shimmering and Mirage*. IEEE Transactions on Visualization and Computer Graphics, 13:179–189, 2007.

[320] Zhao, Y., Wang, L., Qiu, F., Kaufman, A. und Mueller, K.: *Melting and Flowing in Multiphase Environment*. Computers Graphics, 30:2006, 2006.

[321] Zhu, H., Liu, X., Liu, Y. und Wu, E.: *Simulation of miscible binary mixtures based on lattice Boltzmann method: Research Articles*. Computer Animation and Virtual Worlds, 17:403–410, 2006.

Index